PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/44 — A1

(11) International Publication Number: WO 98/57260

(43) International Publication Date: 17 December 1998 (17.12.98)

(21) International Application Number: PCT/US98/12050

(22) International Filing Date: 9 June 1998 (09.06.98)

(30) Priority Data:

60/050,055	13 June 1997 (13.06.97)	US
08/903,896	31 July 1997 (31.07.97)	US

(71) Applicant: TRUE SOFTWARE, INC. [US/US]; 300 Fifth Avenue, Waltham, MA 02154 (US).

(72) Inventor: STANKIEWICZ, Michael, W.; 25 School Street, Townsend, MA 01469 (US).

(74) Agent: SOUTHWORTH, J., Scott; Testa, Hurwitz & Thibeault, LLP, High Street Tower, 125 High Street, Boston, MA 02110 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: SYSTEMS AND METHODS FOR SCANNING AND MODELING DEPENDENCIES IN SOFTWARE APPLICATIONS

(57) Abstract

A software application change management system uses a scanning system, and information model processor, and a release system to dientify dependencies among application files for releasing the application to end–users. The scanning system analyzes dependencies in application files, which can originate from different software development sources, including different software configuration management (SCM) tools as well as vendor supplied application code. The scanning system can include parsers for parsing different types of application files. The information model processor models the dependencies in an acyclic dependency graph, which is stored in an information repository. The information model processor can produce the dependency graph in different output formats for use by different release systems. A release system uses the dependency graph to identify and distribute a release version of the application. The system can also include a report generator which produces reports from the dependency graph.

BNSDOCID: <WO___9857260A1_J_>

BEST AVAILABLE COPY

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

BNSDOCID: <WO___9857260A1_I_>

Systems and Methods for Scanning and Modeling Dependencies in Software Applications

Claim of Priority

This application claims priority to U.S. Provisional Patent Application Serial No. 60/050,055, filed June 13, 1997 entitled "Application Change Management with Dependency Graphs" Docket No. TRU-002PR, and U.S. Patent Application Serial No. 08/903,896, filed July 31, 1997, entitled "Systems and Methods for Scanning and Modeling Dependencies in Software Applications" Docket no. TRU-002, the teachings of which are herein incorporated by reference.

Field of the Invention

The invention relates to systems and methods for managing software application change, and more particularly to information modeling systems that model application dependencies and configurations.

Background of the Invention

Historically, software was developed by one person or a small group of individuals. An application consisted of a few files that were relatively easy to track as they changed. The dependencies and relationships among the files were readily known to the developers.

As software engineering has developed, software configuration management (SCM) has developed to manage changes in software projects. Because software projects have become larger and increasingly more complex, a project usually has many different versions. Typically, version numbers such as 1.0, 2.0, etc. indicate major versions of a software project and 1.1, 2.1, etc. indicate minor versions of a software project. As a software project develops, engineers often find the need to rebuild or reproduce an earlier version. For example, when version 3.0 has been released, there may be a need to reproduce version 2.0 for a special project or a special customer. Different paths or "branches" of the software can also be developed, especially when software runs on different platforms or when specialized versions are needed.

Over time, software projects, whether applications or operating systems, have become very complex, with a large number of files needed in an official "release" version of the software. SCM systems keep track of the different versions of each file, the changes to each file, and which versions of which files belong in a given release. A problem in this area is properly tracking the dependencies among many files. Usually, a software project has one or more source files, which are processed and/or or compiled to produce one or more derived files. The derived files

normally make up the released application that is given to an end user. Often, a software project is expected to run on different hardware platforms, such as INTEL™ based PC computers, UNIX® workstations, or IBM mainframe computers. In this situation, then there can be different source or derived files for each hardware platform. SCM systems must keep track of the different derived files and insure that the correct files are selected for a specified version of the software package for each specific hardware platform. The dependency information often requires a large amount of manual effort to develop and maintain, yet is critical to the success of the software project. The problem is further complicated when different parts of the software package are developed under different SCM system. Each SCM system may have a scheme or framework to help a system administrator track dependency relationships within that particular SCM system. However, it can be difficult to track dependencies across different SCM systems, particularly when each SCM system was originally designed to operate on a particular operating system on a particular hardware platform.

It is therefore an object of the invention to provide systems and methods for automatically scanning dependency information from files developed in different SCM systems.

It is a further object of the invention to provide systems and methods for creating and maintaining dependency information models that are independent of any one SCM or one hardware platform and provide software dependency links among files across different projects, products, and platforms.

It is a further object of the invention to provide systems and methods for providing dependency information to facilitate releasing software to release and distribution systems independently of the particular release and distribution system or hardware platform used.

Summary

The invention achieves the above objects by providing systems and methods for scanning application files, modeling dependencies among the application files using a dependency graph, and providing the dependency graph to release systems for distributing a release version of an application. The application files for a given application do not need to be from any one software source. For example, the application files can be developed under more than one SCM system. The scanning system determines the dependency relationships among the application files and models them in an dependency graph. An information processor provides the

dependency graph in different formats for use by different release systems. A release system then creates a release version of the application software to be released to end-users.

In one aspect, the invention is understood as a system for managing change in a software application. The software application includes application files that originate from a plurality of software development sources. The system includes a scanning system for analyzing the application files to determine the dependency relationships among the files. An information model processor processes the dependency relationships to derive an acyclic dependency graph representing the dependency relationships. The system of the invention also includes a release system that uses the acyclic dependency graph to create a software release package based on the application files.

In one embodiment, the application files are developed using a plurality of SCM tools. In a further embodiment, the software development source for the application files includes vendor supplied application code.

The system of the invention can also include an information repository for storing and retrieving the acyclic dependency graph. The information model processor can produce the acyclic dependency graph in an output format suitable for use by the release system. The output format can be a vendor specific format, a generic format, or the standard Management Information Format (MIF). In a further embodiment, the scanning system can include one or more parsers for parsing the application files to determine the dependency relationships. The system of the invention can also include one or more reference areas in which the application files reside. In one aspect, the system of the invention also includes a report generator for generating reports from the acyclic dependency graph.

In a further embodiment, the system of the invention includes an information repository for storing and retrieving the acyclic dependency graph, a plurality of first computer systems, a second computer system, and a third computer system all connected via a network. The first computer systems include reference areas in which the application files reside and are analyzed by the scanning system. The second computer system has the scanning system, information model processor, and the information repository running on it. The third computer system has the release system running on it.

In another aspect, the application comprises source and derived components, the derived components deriving from the source components. The acyclic dependency graph includes component references and links between the component references. The component references

include references to source components and to derived components. The links include derived links and cross-links. The derived links connect source component references and derived component references. The cross-links connect two derived component references that derive from the same source component.

In one embodiment, the invention can be understood as a method for managing change in a software application. The method includes scanning the application files to determine dependency relationships, processing the dependency relationships to determine an acyclic dependency graph, and creating a release version of the software application using the acyclic dependency graph. The scanning step can include parsing the application files with one or more parsers. The scanning step can also include accessing one or more reference areas holding application files.

In another aspect, the method can include storing the acyclic dependency graph in an information repository and retrieving the dependency graph from the information repository to create a release version of the software application. The method can also include producing the acyclic dependency graph in an output format for use by a release system. The method of the invention can also include generating reports from the acyclic dependency graph.

Brief Descriptions of the Drawings

FIG. 1 illustrates a high level view of an application change system according to the invention;

FIG. 2 illustrates in a functional block diagram an application change system according to one embodiment of the invention;

FIG. 3 illustrates a network of computer systems executing the system depicted in FIG. 2;

FIG. 4 illustrates in a functional block diagram a further embodiment of the invention depicted in FIG. 2;

FIG. 5 depicts a flow chart for one embodiment of the invention illustrating the reference area scanning process;

FIG. 6 depicts a flow chart of the project scanning process referred to in FIG. 5;

FIG. 7 depicts a flow chart of the product scanning process referred to in FIG. 5;

FIG. 8 depicts a sample dependency graph of the type used with one embodiment of the invention;

FIG. 9 depicts another sample dependency graph of the type used with one embodiment of the invention;

FIG. 10 illustrates a high-level overview of the classes used in an information model for one embodiment of the invention;

FIG. 11 depicts the classes in a dependency submodel of the information model illustrated in FIG. 10;

FIG. 12 depicts the classes in a release submodel of the information model illustrated in FIG. 10;

FIG. 13 depicts the classes in a change request submodel of the information model illustrated in FIG. 10;

FIG. 14 depicts the classes in a user privileges submodel of the information model illustrated in FIG. 10;

FIG. 15 depicts the product component classes used in the information model illustrated in FIG. 10; and

FIG. 16 depicts a process server approach used in a further embodiment of the invention illustrated in FIG. 4.

Detailed Description of the Invention

The illustrated embodiments of the invention can be understood as systems and methods for managing software application change. The embodiments of the invention relate particularly to information modeling systems that model application change and configuration during the software application development process.

FIG. 1 illustrates a high level view of one embodiment of an application change management system 10. The application management system 10 receives input files 18 from different software development sources, which can include those produced with a vendor specific SCM (software configuration management) tool 12 or a generic SCM tool 14, such as UNIX SCCS. The input files 18 can also include vendor supplied application code 16, which can include executable and other files provided with a vendor's application. The application change management system 10 scans the dependencies in the input files 18 and constructs an information model, which is used to produce an output configuration in different formats. These formats can include a generic format 22, a standard Management Information Format (MIF) 24, or a vendor specific format 26. This output configuration is used by software distribution tools

20 to construct software release packages for the software application files scanned as input 18 to the application change management system 10.

Components Overview

FIG. 2 illustrates a functional block diagram of the application change management system 10. The system 10 includes a scanning system 28, an information model processor 30, an information repository 32, a release system 34, a report generator 36, and a reference area 38.

The reference area 38 contains directories and application files that serve as input into the scanning system 28 and the rest of the system 10.

The reference area 38 can contain project, make, derived, source and include files. To scan any file, the scanning system 28 assumes it to be in the reference area 38. Alternately, the scanning system 28 can be given a specific project file for scanning, in which case the reference area 38 is set to the location of the project file.

The reference area 38 typically contains the files that make up one or more software applications. These files have been developed by one or more software engineers, usually in conjunction with any of a number of commonly used SCM systems, such as TRUEchange™ of True Software Inc. of Waltham, Massachusetts or ClearCase® of Pure Atria Inc. of Cupertino, California. The reference area 38 can contain files developed under more than one SCM system. That is, the embodiment of the invention does not require that all files in the reference area 38 be developed in conjunction with a single SCM system. Alternatively, more than one reference area 38 can be scanned during the scanning process for one application or software project. For example, the scanning system 28 can scan two reference areas 38, each containing files developed under a different SCM systems.

The scanning system 28 scans and analyzes the files in the reference area 38 to conduct an analysis of dependency and interconnection relationships among the files. The scanning system 28 does the dependency analysis for a given project or product by first scanning the project files or makefiles. Then source files are scanned if required. The scanning system 28 scans for the following dependency information:

- derived to derived file dependencies
- source to derived file dependencies,
- source to source file dependencies,
- include file dependencies,

- external project-product dependencies,
- file size, time stamp, version, etc.

The scanning system 28 stores the above dependencies inside a dependency information object submodel in an information repository 32, which can be later retrieved either to generate reports or to be used by a release system 34. The dependency information submodel is described in detail in connection with FIG. 11. Given a reference area 38, the scanning system 28 can scan the files for all the projects in the reference area 38 for dependencies. The scanning system 28 can be used to either scan all projects in the product, one project in the product, or all the projects and products in the reference area 38. The scanning system 28 reports any error during scanning.

The scanning system 28 can include one or more parsers 68 (see FIG. 4) to analyze the relationships among the files. For example, the scanning system 28 can include a Visual Basic parser to scan files developed using Visual Basic, a Visual C++ parser to scan files developed using Visual C++, and an SQL parser to scan scripts and files for SQL statements. The scanning system 28 can include other parsers 68 to scan files in other programming and script languages used to develop software applications or projects.

The information model processor 30 processes the scanned dependency information into an information model including a dependency submodel that tracks the dependency relationships among the scanned files in one or more dependency graphs. When the parser 68 identifies any dependency while scanning, it creates the respective object to store the dependency relationship. Hence, when the scanning is complete, the dependency information model has all the dependency relationships. The dependency information model is discussed in more detail below.

The information repository 32 is used to store the output dependency information from the information model processor 30. The scanning system 28 has the flexibility to either store the dependency information in the information repository 32 or discard it, after the scanning is complete.

The report generator 36 produces a report of the dependency information as modeled by the information modeling processor 30. Once the scanning is complete, the scanning system 28 generates a report by invoking the report generator 36. The report generator 36 also provides a set of queries to extract any specific dependency information stored in the information repository 32. After scanning, when the report is generated, the user can choose to either store this dependency information in the information repository 32 or discard it. Based on the report from

the report generator 36, the user can manually add additional dependency relationships that the scanning system 28 did not provide.

The information model processor 30 can provide a configuration model to a release system 34 that can then produce a release version of the software application based on the configuration model stored in the information repository 32. The information model processor 30 can produce output in a format acceptable to any of a number of commonly used release and distribution systems, including TRUErelease™ from True Software, Inc., or the standard MIF format used by Tivoli Systems (IBM) of Austin, Texas.

The system 10 depicted in FIG. 2 can reside and execute on one computer system. Alternatively, one or more of the components of the system can reside on separate computer systems interconnected in a network. In one embodiment, the components of the depicted system 10 are objects in a distributed object system, which would allow each object to be placed readily on the same or different systems. It will be understood by one of ordinary skill in the art how to connect the software components of the invention in a distributed network of two or more digital data processing systems. (See also FIG. 3.) Such a distributed system can be constructed using the CORBA™ standard or the Java Bean API for distributed software environments.

FIG. 3 illustrates a network of computer systems used with one embodiment of the invention. Software engineers develop an application, or a new version of an existing application, using one or more software development workstations 40. As the files in the application are developed, they are placed in a reference area 38 on one or more SCM computer systems 42 running SCM software. The SCM computer system 42 is connected through a network 50 to an ACM computer system 44 and one or more release and distribution computer systems 46. A scanning system 28, information model processor 30, and information repository 32 run on the ACM computer system 44. Software release systems 34 run on the release and distribution computer systems 46. The release and distribution computer systems 46 are connected through a network 50 to end-user computer systems 48. The SCM computer systems 42, ACM computer system 44, and release and distribution computer systems 46 work together to develop a release version of the application software, which the release and distribution computer system 46 then releases (sends or makes available over the network) to the end-user computer systems 48, where end-users can obtain and run the new version of the application software.

FIG. 3 shows only one embodiment of the invention. The software of the invention can alternatively be set up to run on alternate combinations of physical computer systems. For example, the scanning system 28 and release systems 34 can execute on one computer system. The reference area 38 can also be placed on the same computer system. In an alternate embodiment, a software development workstation 40 can both run SCM software and have its own reference area 38.

Scanning System and Process

FIG. 4 is a functional block diagram illustrating a further embodiment of the invention. The embodiment comprises the scanning system 28, the information model processor 30, the information repository 32, and the release system 34. The scanning system 28 comprises a report generator 36 for producing reports on the dependency relationships, a source reference area 38 that holds application files to be scanned, a scanner 70 including one or more parsers 68 for parsing application files in the reference area 38, and a scanning system GUI 72 that an operator uses during the scanning process to monitor the process, add additional dependencies manually, generate reports using the report generator 36, and perform other functions. The release system 34 includes a release controller 60, a release report generator 62 that produces reports about the released version of an application, a release GUI 64 used by an operator monitoring the release process, and a release staging area 66 that contains files ready to be released for a specific release version of an application.

The scanning system 28 is designed to provide plug and play capability for different parsers 68 for different application components. For a given language, one parser 68 can scan the project file or makefile, and another parser 68 can scan the source files. For different types of source files there can be different parsers 68. The scanning system 28 determines which parser 68 to invoke depending on the file type.

The scanning system 28 can scan:

- a project by analyzing project, make or source files
- a product, and hence all the projects within the product
- a reference area 38 and hence all the product and projects within that reference area 38.

The scanning schemes for the above scenarios are explained in more detail later.

When a project is scanned for dependencies, the scanning system 28 scans the project file

or makefile for that project and identifies dependency information, such as dependencies from the source files, include files, and derived files. The scanning system 28 also identifies any external dependencies such as which external derived file or external project the project is dependent on. Other information that is in the makefile or project file is also captured. The source file can be further scanned to find any other derived file dependency that is there. Based on the source file dependency just captured, the scanning system 28 further decides whether to scan other source files. If necessary, it then invokes the respective parser 68 to scan the other files. Once all the dependency information is captured, the scanning system 28 invokes the report generator 36 to generate the report. The user can choose to store this information in the information repository 32 or discard it. This completes the scanning process. (See FIG. 5 for more detail on this process.)

If the scanned dependency relationship is stored, then it can be used later to generate dependency reports that can be used by a release and distribution system 34. In one embodiment of the invention, if a project whose dependency information is already in the information repository 32 is rescanned then the dependency information is overwritten. All relevant files are rescanned instead of incremental scanning only those files that have changed.

In one embodiment, the scanning system 28 uses Java Compiler (JavaCC), a Java parser generator to implement the parser 68. JavaCC provides a combined lexer/parser generator. It allows use of regular expressions to express lexical and syntactic patterns. JavaCC is also flexible without any entry points. Any nonterminal point can be chosen as a start point. The scanning system 28 requires parsers 68 for each source file and project file representation to support scanning source files and project files in a particular language.

In one embodiment, the scanner 70, the report generator 36, and the parsers 68 are all identified as objects implemented as Java Beans. The scanner object 70 fulfills the control and manager role for the entire scanning process. The scanner 70 controls the parser 68 and also the query and report generator module 36. The parser 68 being an object, different parsers 68 can be used in a plug and play approach. The scanner 70 can have one or more relationships to one or more parser objects 68. Having different parsers 68 is accomplished by having different instances of the parser object 68.

The scanner 70 can scan a project, a product or an entire reference area for dependency information. The scanning scheme for each scenario is explained below. Scanning a project is

part of scanning the product, and scanning a product is part of scanning the entire source reference area 38.

The scanning scheme for scanning an entire reference area 38 is illustrated in FIG. 5. First, the scanner gets all the products associated to the reference area (step 200). If any products are found (step 202), then for each product, the scanner invokes product scanning (step 204, see also FIG. 7). If not, then the scanner finds all the project files or makefiles in the reference area 38 (step 206). For each of these files, the scanner invokes project scanning (steps 208 and 210, see also FIG. 6). The dependency information produced by scanning is saved in the information repository 32 depending on the choice made by the user to save or discard the information (steps 214 and 216).

If the reference area 38 is not associated to any product, then all the projects in that reference area 38 are searched and scanned for dependency information.

The scanning scheme for scanning a project for dependency information is illustrated in FIG. 6. The input to the scanner can be either a project name, reference area, a project file, or a makefile. If the scanner 70 receives a project file or makefile as input, the scanner 70 invokes a parser 68 to scan the file (steps 220 and 228). If the scanner 70 receives just the project name and the reference area 38, then the scanner 70 gets the project object with that name from the information repository 32, and determines the location of the project file from the location information in the object (step 222). If the project object does not exist, then the scanner 70 searches the reference area 38 for a project file where the executable name is the project name (step 222). If the scanner 70 does not find such a file, then the scanner 70 reports it as an external project and stops scanning (steps 224 and 226). If such a file is found, then the scanner 70 invokes scanning the file (step 228).

The scanner 70 invokes the respective parser 68 to scan the project file or the makefile depending on the file type (step 228). For example, for a VB (Visual Basic) project file the scanner 70 invokes the VB project file parser, or for a VC++ (Visual C++) makefile the scanner 70 invokes the VC++ makefile parser. The parser 68 scans for the dependency information and creates this relationship information in the dependency information model in the information model processor 30. The parser 68 identifies source file dependencies, derived file dependencies, and other dependency information in the project file or makefile. For each source file identified in steps 230 and 232, the scanner 70 invokes the respective parser (step 228) if that

file is required to be scanned (as determined by file type). The parser 68 stores the derived file dependencies if any.

For each derived file identified in step 234, the project dependencies are classified, if there are any (step 236). The derived file is checked to determine if it is a project in the reference area 38. If yes, then the dependency is classified as a project-project or a project-product dependency. The dependency may be classified as a project-project dependency if both projects should belong to the same product. Otherwise the dependency is classified as a project-product dependency. If none of the above, then the dependency is classified as a project-derived dependency where the derived file is an external derived file (system or some external product).

This completes the scanning of a project, and the scanner 70 invokes the report generator 36 to generate a report (step 238). If the user chooses to save the dependency information, then it is saved in the information repository 32.

For product scanning, it is assumed that the product object by that product name exists in the information repository 32 and has a list of projects associated with it. If not, then the user should add the list of projects to the product first and then invoke the scanner 70. The scanning scheme for scanning a product is illustrated in FIG. 7. The scanner 70 receives the name of the product and reference area 38 as input. The scanner 70 gets the product object from the information repository 32 if it exists (steps 240). The scanner 70 then gets the list of projects associated to this product (step 242). For each project, the scanner 70 uses the reference area information and starts scanning the project using the project scanning scheme shown in FIG. 6 (step 246). If the product object does not exist, then an error is reported and scanning stops.

Completing the scanning of all the projects completes the scanning of the product. The user can now choose whether to save this dependency information in the information repository 32 or discard it.

Relationship information can be obtained once a project is scanned by using a querying scheme such as the one below.

QUERY INPUT	AVAILABLE OUTPUTS
Project name	Source file dependency
File name	Projects that use this file (or are dependent on this file)
Project name	Derived file dependencies
Derived file name	Projects dependent on this file
Project name	Project dependencies
Project name	Project which is dependent on this project
Project name	Product dependencies
Project name	The dependency information in a report form
Project name	Include file dependencies
Product name	Project-project dependencies
Product or project name	The external dependencies
Derived file name	File size and date, time stamp

The dependency model which stores the dependency information provides the interface for the navigation of the dependencies. The report generator 36 interacts with the dependency model to fetch the information to satisfy these queries.

Any information other than the stated dependencies that are identified in the project file or makefile is modeled to be stored in an object hierarchy or the dependency object model. This makes it flexible for a user to use, navigate, and query this set of information later.

Information Model

The information model processor 30 uses an information model to construct acyclic dependency graphs that model the dependency relationships among application files.

FIG. 8 depicts a sample dependency graph 100. The dependency graph 100 maintains relationships and cross-links among source and derived components. The acyclic dependency graph is the deriving structure for all operations. The graph includes source or derived component types (nodes) and the relationships between nodes (links). The source or derived component types can include files, objects, or other software components. The dependency graph also includes roles associated with nodes and roles associated with links. The node/ link combination provides a broad nature relationship with group aggregations, and the detail relationships at file level, as needed.

For example, application files can include source files and derived files, which are represented as source file references and derived file references in the dependency graph. These file references are then connected by derived links and cross-links in the dependency graph.

FIG. 8 shows a sample dependency graph 100 for COBOL files. The three nodes in the dependency graph 100 shown in FIG. 8 are

- source component: COBOL source file 102
- derived component: COBOL object file 106
- derived component: DBRM file 110 (preprocessor created derived component for separation of embedded SQL stat)

FIG. 8 shows the links in the sample dependency graph 100. Link 104 is a derived link which must be derived with a compiler whenever COBOL source files are changed. Link 112 is a derived link that must be derived with a preprocessor whenever COBOL source files with embedded SQL statements are changed. Link 108 is a cross-link that must maintain a linkage

between the DBRM file 110 and the COBOL object file 106, such that a new DBRM file 110 is associated with a new COBOL object file 106 if both Links 104 and 112 are changed.

FIG. 9 shows a sample dependency graph 120 using Visual Basic (VB) files. The three nodes in the dependency graph 120 shown in FIG. 9 are

- VB source file 122 with embedded SQL statements that query a database
- Database table definitions 126
- Stored procedures 130

FIG. 9 shows the links in the sample dependency graph 120. Link 124 is a link between the VB source file 122 and the database table definitions file 126. Link 132 is a link between the VB source file 122 and a VB stored procedures file 130. Link 128 is a link between the database table definitions file 126 and the VB stored procedures file 130. The files are independent, that is, they are not derived from each other. However, the files are linked in that a change in one file indicates a new version of that file that may require a new version of one or both of the other files.

The graphs can also include links to software components across multiple hardware platforms, to software components developed using different SCM systems, and to vendor-supplied application code.

In one embodiment of the invention, the dependency graph approach is used by the information model processor 30 using an information model illustrated in FIGs. 10-15. These figures use the UML (Uniform Modeling Language) notation to show the classes in the information model and their relationships. FIG. 10 depicts an overview of the classes in the information model. The classes include ProductComponent 300, RepositoryProject 302, Repository 304, Version 306, User 308, ChangeSet 310, File 312, DerivedFile 314, Project 316, DependencySpec 318, VBProject 368, and CPProject 370.

A ProductComponent 300 represents a software product which may stand on its own or be included in another ProductComponent 300. A ProductComponent 300 which is change managed has one or more RepositoryProjects 302. There may also be ProductComponents 300 which have no RepositoryProjects 302. An example of a ProductComponent 300 which has no associated RepositoryProject 302 is a third party product which is packaged as part of a release. A ProductComponent 300 can have dependencies on other ProductComponents 300. Each

ProductComponent 300 has a unique key based on its name, majorVersion, minorVersion, and Revision.

A RepositoryProject 302 represents a SCM system project. Each Repository Project 302 has a unique key based on its name and the Repository 304.

A Repository 304 represents an SCM system repository. It is the physical location for the source files. Each Repository 304 has a unique identifier based on its name.

A Version 306 represents a single version of a RepositoryProject 302. It contains one or more Projects 316. Each Version 306 has a unique key based on its name and the RepositoryProject 302.

A User 308 represents a registered user of the application change management software. Each user has a unique key based on a name.

A ChangeSet 310 is a set of files containing related software changes. A ChangeSet 310 contains code changes in response to a single change order. However, a ChangeSet 310 may not contain all of the changes required. Multiple ChangeSets 310 can be associated with one change order. Each ChangeSet 310 has a unique identification key.

A File 312 represents a source application file, which has a modification date. A DerivedFile 314 represents a derived application file, which has a creation date indicating when it was derived from the source file. If the modification date of a File 312 is more recent that the creation date of a DerivedFile 314, then the DerivedFile 314 needs to be recreated. That is, the DerivedFile 314 needs to be derived again from the source File 312.

A Project 316 is a file group. Files within a RepositoryProject 302, versions are organized into projects. Each Project 316 has a unique key based on its name and the Version 306.

DependencySpec 318 is the super class for the dependency information model. All the classes for different dependency types inherit from this super class. Some of the methods are common to the subclasses (e.g., store in the information repository) and some are different (e.g., different queries).

VBProject 368 represents a Visual Basic project. CPProject 370 represents a C++ project developed using Visual C++.

In one embodiment of the invention, the information model supports the Common Information Model (CIM) of the Desktop Management Task Force (DMTF). The relevant part

of the CIM model is the Application Schema Definition described by the Application Management Working Committee of the DMTF.

In another embodiment, the information model also provides support to generate a MIF file to support integration with desktop management software. In a further embodiment, the information model supports AMS (Application Management Specification) files that can be read in by Tivoli's Developer Toolkit and imported into Tivoli's Software Distribution Toolkit.

Dependency Information Submodel

The dependency information submodel is designed to support storing of dependency relationships and easy retrieval of the same. FIG. 11 illustrates the dependency submodel. The dependency information is a relationship between two objects. It is modeled as a separate association class instead of loading both the objects to which the association is related. In addition, methods specific to each dependency relationship can be placed in each separate class. Further, based on the type of association (for example, project to project, project to derived, etc.) the dependency model is modeled to store different types of relationships. All the different type of dependency classes inherit from a super class, DependencySpec 318. In a further embodiment, the model is flexible enough to support user defined dependencies to be stored and retrieved. The classes in the dependency information model are shown in FIG. 11.

ProjFileDep 320 stores the dependency relationship between a project and a file in an object of this association class. A project can contain one or more files. For example, Project A contains source files x, y, z. A file can be used by more than one project (for example, a form that is shared between projects).

ProjProdDep 322 stores the dependency relationship between a project and a product in an object of this association class. A project to derived file dependency gets converted to ProjProdDep 322 if the derived file is identified as one of the projects in the reference area 38 but to a different product. In a further embodiment of the invention, this class is used to store user defined external product dependencies like ORACLE, Sybase, PSE/PRO, Rogue wave, etc.

DervFileDep 324 is used by the dependency model to store information such as a derived file that needs some input file or source file for installation (or needs to be packaged for release) in an object of this class. Though this information is not deductible through scanning, this is modeled to support user defined dependencies.

ProjProjDep 326 is used to store the dependency relationship between a project and a project in an object of this association class. Actually, a project to derived dependency gets converted to ProjProjDep 326 if the derived file is identified as one of the projects in the reference area 38 (meaning a different project of the same product). For example, a user interface project uses the core client derived file, which by themselves are projects.

ProjDervDep 328 is used to store the dependency relationship between a project and a derived object in an object of this association class. For example, Project A is dependent on an external derived file "olepro32.dll". A project may be dependent on zero or more derived files. A derived file can be used by one or more projects.

Class DervDervDep 330 is used to store the dependency relationship between a derived file and a derived file in an object of this association class. This type of relationship may be found in an environment where a derived file that was built internally within a project is not treated as a separate project (typical in a UNIX environment).

A ReferenceArea 332 represents the physical location of all the source and object code for a ProductComponent 300. Each ReferenceArea 332 has a unique key based on its name, the Platform 350, and the ProductComponent 300.

The other classes identified in the scanning process are Scanner 334, Store Dep/Info 336, Parser 338 and GenerateReport 340. In one embodiment of the invention, there is only one instance of a Scanner object 334. It can have one or more Parser objects 338. The Scanner object 334 is also associated to the object ReferenceArea 332 for the purpose of scanning. The Scanner 334 is associated to the GenerateReport 340 object. GenerateReport 340 object also handles query functionality. Alternatively GenerateReport 340 can be termed a Report Generator.

The Scanner object 334 has interfaces to control and manage the entire scanning system process. In one embodiment of the invention, the scanning process can be made simpler by utilizing a process engine architecture with a process server 74 and an information process model 76 (see FIG. 16). Even updating the dependency information in the information repository 32 when a file gets deleted can also be modeled as a process engine object with rules and events.

The dependency information is modeled and designed such that the objects which share a relationship are not loaded with this information in both the classes. Instead the dependency relationship is stored in a separate association class. This also makes the information storage or

retrieval much easier. The relationship information is stored just with object names (just as strings) and not objects themselves, which gives the best performance for both storing, updates, and retrieval of data. This approach reduces overhead maintained with object references. This approach also provides two way direction for information access without much overhead.

Release Submodel

The release submodel is illustrated in FIG. 12 and shows the release and the relationship of the release to other objects. The classes in the release submodel not described previously are discussed below.

VersionedObject 342 is a base class for Release 344 and ProductComponent 300. It supports the versioning behavior for Releases 344 and ProductComponents 300.

A Release 344 represents a product to be "shipped". It is defined as a set of product components which could represent applications, libraries, scripts, or other files, i.e. any collection of files to be deployed. A release is defined and managed by the release manager. For each release there is one or more target platforms and one or more release areas. Each Release 344 has a unique key based on its name, majorVersion, minorVersion, Revision, and build.

A FilePackage 346 is a list of files to be included in a release. A FilePackage 346 contains all of the files for a specific platform release of a product component. If the product component has subcomponents, the file package will include a file package for each subcomponent. A FilePackage 346 corresponds to a Tivoli file package. Each FilePackage 346 has a unique key based on its name, ProductComponent 300, and Platform 350.

A Platform 350 represents a system supported by a ProductComponent 300. A ProductComponent 300 runs on one or more Platforms 350. Each one has one or more ReleaseAreas 352 and one or more ReferenceAreas 332. A Release 344 has one or more target Platforms 350. Each Platform 350 has a unique key based on its name.

A ReleaseArea 352 represents the physical location of those files (executables, libraries, scripts, etc.) which are shipped as part of a release. Files in the ReleaseArea 352 are copied from a ReferenceArea 332 in preparation for release. Each ReleaseArea 352 has a unique key based on its name, the Platform 350, and the Release 344.

A moduleLocation 354 represents the location of a software module in a ReferenceArea 332, and moduleDestination 356 represents the location of a software module in a ReleaseArea 352.

A FileGroup 358 represents a group of related source Files 312 and DerivedFiles 314.

Change Request Submodel

The change request submodel shows the interaction between the objects related to problem tracking and reporting (see FIG. 13). The classes in the change request submodel not described previously are discussed below.

A CallIncident 362 represents an incident report. One or more change requests are associated with one change order. Each incident report has a unique identification key.

A ChangeOrder 360 requests a software change. It contains detailed information describing a modification of a software product. A ChangeOrder 360 is in response to one or more CallIncidents 362. Each ChangeOrder 360 has a unique identification key.

User Privileges Submodel

The user privileges submodel describes the model for specifying which users are allowed to perform certain tasks (see FIG. 14). The classes in the user privileges submodel not described previously are discussed below.

Each User 308 may have one or more Roles 364 assigned to them. A Role 364 defines a collection of tasks that can be performed by any user who is assigned that role. The set of tasks associated with the role defines the responsibilities of the role. Tasks may be system or user defined. Four roles are system defined. They are Administration, Developer, Project leader, and Builder. Each role has a unique key based on its name.

A Task 366 represents a single logical operation to be performed. A Task 366 may be included in more than one role. Tasks are system defined. Each task has a unique key based on its name.

ProductComponent Classes

The ProductComponent 300 classes (see FIG. 15) are discussed below, except for ProductComponent 300 itself, which was discussed for FIG. 10.

An INIFileEntry 372 represent an initialization file entry, if a ProductComponent 300 optionally has an initialization file associated with it. The initialization file can be an application initialization file or one of the pre-existing system initialization files, such as those associated with the operating system or Microsoft® Windows®. A RegistryEntry 374 represents an entry

for the ProductComponent 300 in a registry, such as a Windows® 95 or NT Registry. SpaceRequirements 376 indicates the space required for a ProductComponent 300. A Signature 378 represents a uniquely identifying signature for a ProductComponent 300.

Information Process Model

In one embodiment of the invention, as shown in FIG. 16, the information model is a process object model that tracks application components, relationships, and dependencies. FIG. 16 is similar to FIG. 4, but includes a process server 74 and an information process model 76 in place of the information model processor 30 of FIG. 4. Component dependencies information obtained either through scanning or direct user inputs are placed into the information repository 32. Writing and reading of process model data based on the information process model 76 into and out of the repository 32 is serviced by the process server 74. The information repository 32 can also be termed a configuration repository.

The release system 34 provides a workflow model and functions that developers, project administrators, release managers and builders use to configure and manage their development build/release in a consistent manner. A development release can be comprised of one or more application products on multiple operating system platforms. The release system 34 uses the information process model 76 and a set of object services to manage and service all of the supported functions. The release configuration data is based on the information process model 76 and stored in the information repository database 32. Users can use the stored information to generate contents for DMTF based MIF standard format files including those supported by Tivoli. The information model's API object services support or are made available to the release system GUI client front end 64 or other third vendor clients supported by a distributed process server. In one embodiment, the release system 34 is TRUErelease from True Software, Inc.

The release system 34 includes a standalone GUI desktop front end 64 that supports a set of functions to create and manage a release configuration and generate reports. The release system 34 also has the ability to manage a source code reference area 38 and build a release staging area 66 in coordination with an SCM system. When used in conjunction with a scanning system 28, it also allows users to apply the scanners 70 to extract application product component dependencies for a release configuration. When used in conjunction with a tracking system, an SCM change system, and a scanning system 28, users can manage a configuration release from

the start of a problem call, turning it into change request, going through development and testing, all the way through the development life cycle to the software or software patch release phase.

In one embodiment, the GUI desktop front end 64 for the release system 34 is built with Java for a user to create one or more release configurations. Users can define and freeze (checkpoint) a configuration, assign a reference area path for each release platform and perform code pull from a SCM system (such as TRUEchange from True Software, Inc. or PVCS from INTERSOLV, Inc. of Rockville, Maryland). Users can bring up and view a set of configuration reports.

The scanning system 28 can be run as a standalone software system or in conjunction with a release or SCM change system. Running standalone, it provides a window desktop GUI 72 from which users can extract component dependencies from multiple application projects by scanning and/or using user inputs. The set of dependencies and relationships information can be derived by scanning a number or group of file sources including software application source code, project files, and makefiles. The output or the scanning information is then stored into an information repository database 32 in the context of the dependency information submodel. Users can view the result details inside a number of dependency text reports, and in a further embodiment, be able to view and navigate between these dependent components in a graphical display or dependency tree diagram. In one embodiment, the scanning system 28 is TRUEimpact from True Software, Inc.

The scanning system 28 can be used by project administrators and builders as well as developers. When used together with a release system 34, the set of development life cycle application dependencies can be retrieved from the repository to provide an information set that can be used for application release management. Examples include makefile generation for building applications, and generation of DMTF based MIF files that are useful for release distribution and deployment. MIF standard files are supported by vendor tools like TME 10 Software Distribution (Courier) from Tivoli.

In one embodiment, any number of scanners 70 can be packaged with a scanning system 28. A user can associate one or more scanners 70 to an application product. In one embodiment, the scanning system 28 can use artificial intelligence to detect the presence of source files that are appropriate for scanning. The parser 68 for the scanner 70 can be built using JavaCC, and a set of grammar rules can be developed for each scanner 70 to track a program or application

product's component dependencies. Instead of applying scanning during check-in from a change system, all scanning is performed against a source reference area 38 right after a refresh or any other time to record or update the latest dependency information.

In one embodiment, the scanning system 28 provides a GUI desktop front end 72 built with Java for a user to selectively scan a set of application project related files residing in a designated build reference area 38 and/or staging areas in the user's work space. Users can bring up and view dependency text reports. In a further embodiment, users can view and navigate the application's component dependencies in a number of graphical diagrams using an interactive graphic display system.

Each source path that has been subjected to scanning and each application project that has been subjected to scanning are remembered by the tool. The report generator 36 produces a detailed report after each scanning to show the application and file modules that have been processed with clear identification of any inconsistency and exceptions. The scanning system 28 has plug and play support for a variety of application scanners 70 including those developed by the customers themselves.

In one embodiment of the invention, there is intelligent integration between the scanning 28 and release system 34. If a scanning system 28 is installed, release system 34 users have the option to invoke different types of scanners 70 against a release reference area pointed to by the release configuration. The release system 34 can also share information with the dependency information submodel data from the scanning system 28 in the information repository 32.

In order for each system to recognize the presence of each other, under a Windows® platform, both the scanning GUI client 72 and release GUI client 64 each have a registry entry that indicates that the client is installed and its location.

Users using an SCM change system, release system 34, and scanning system 28 together can generate reports on application changes between project versions, and component dependencies information in a release configuration.

Process Server

In one embodiment, as shown in FIG. 16, the information model is a process object model that tracks application component relationships and dependencies. Component dependencies information obtained either through scanning or direct user inputs are placed into the information repository 32. Writing and reading of process model data based on the

information process model 76 into and out of the repository 32 is serviced by the process server 74. The information repository 32 can also be termed a configuration repository.

In another embodiment, the information model includes a composite set of Java class objects based on Sun's Java Bean component architecture. The object information model must be able to track a variety of application project components coming from different development tools and environments like Visual Basic, Visual C/C++ and Java. In other words, the model is extensible and can support different types of applications built from different programming language and tools on different operating system platforms. The information model also supports and tracks information that can be applied to the CIM model of the DMTF. Under CIM, applications are made up of components where their relationships are tracked MIF format definitions. By tracking and storing information supported, the scanning system 28 helps provide the users with the means to generate the MIF standard format file common to all tools that support the MIF standard.

In the embodiment illustrated in FIG. 16, the process server 74 and the information repository 32 reside on a host server and can service the scanning system desktop client 72 as well as other clients such as the release system client 64 through the exposed API of the process server 74 for the scanning system process. The scanning system GUI desktop client 72 is developed primarily as a thin client. The process server 74 takes advantage of both Java Remote Method Invocation (RMI) and (ORB/CORBA) Object Request Broker technology to service all client requests. The underlying server protocol is transparent to all clients.

In a further embodiment, processes in an SCM change system are provided by multiple process object models serviced by a distributed process server 74. Clients of the process server 74 locate and query these remote process objects on the process server host to represent them visually to the user, as well as invoke specific methods of these objects to perform specific tasks. Each object can have multiple presentations for multiple purposes. For example, the client UI presentations communicate directly with a process object such as the project version object and act on an interface process object such as the checkout object to display project version data on the screen for the user to interact with, and to perform checking out of files.

The distributed process server's architecture achieve the following benefits and high level abstractions. The process server 74 and a process server engine are implemented in Java and quickly portable to any machine running Java VM. The process server 74 supports objects

distribution using either standard CORBA ORB or Java Remote Method Invocation (RMI). The intricate details of their communication and implementation are abstracted out into a single client service interface that make the communication protocol transparent to the client users.

One embodiment of the invention, through a Java ORB server, can support different client program implementations and is independent of the Java language implementation of the process server 74. CORBA 2.0 defines Inter Operable Object references (IOR) that vendors uses to pass object references across heterogeneous ORBs. ORBs provide the same language binding to an object references for a particular programming language. Clients see the object interfaces through the perspective of that language mapping or binding. Clients should be able to work without any source changes on any ORB that supports the language binding.

This support of distributed object programming allows objects to be instantiated and distributed on a separate and remote server host machine which the clients communicate with. These CORBA ORB and RMI based Java server objects are available and published for use over a network on a permanent basis.

The Java RMI server provides an alternative and lighter weight approach to distribute objects for Java (only) based clients unlike ORB. As object distribution technology evolves, in further embodiments, unrestricted protocol changes can be made within this plug and play architecture i.e. underneath this client interface abstraction.

CORBA ORB provides a scaleable server-to-server infrastructure. The process objects can run on multiple process servers to provide load balancing for incoming client requests. It also provides a multithreaded environment for multiple clients. For example, in Java RMI, all the remote calls invoked from the same client may or may not be running on different threads. But the remote calls from different clients are guaranteed to execute on different threads on the process server. The server objects can also act in unison using transaction boundaries and related CORBA services. A well designed process object built on the CORBA services allows the use of the built in concurrency control and transaction services to maintain the integrity of the object's state.

Having described the preferred embodiments of the invention, it will now become apparent to one of skill in the art that other embodiments incorporating the invention may be used. These embodiments should not be limited to disclosed embodiments but rather should be limited only by the spirit and scope of the following claims.

CLAIMS

What is claimed is:

1. A system for managing change in a software application comprising:

a software application which includes application files originating from a plurality of software development sources;

a scanning system for analyzing the application files to determine dependency relationships among the application files;

an information model processor in communication with the scanning system, the information model processor processing the dependency relationships among the application files for deriving an acyclic dependency graph representing the dependency relationships in the application files; and

a release system in communication with the information model processor for using the acyclic dependency graph to create a software release package based on the application files.

2. A system as claimed in claim 1 wherein the application files are developed using a plurality of software configuration management tools.

3. A system as claimed in claim 1 wherein one of the software development sources comprises vendor supplied application code.

4. A system as claimed in claim 1 further comprising an information repository in communication with the information model processor for storing and retrieving the acyclic dependency graph.

5. A system as claimed in claim 1 wherein the information model processor produces the acyclic dependency graph in an output format for use by the release system.

6. A system as claimed in claim 5 wherein the output format is one of a vendor specific format, a generic format, and a Management Information Format (MIF).

7. A system as claimed in claim 1 wherein the scanning system comprises at least one parser for parsing the application files to determine the dependency relationships.

8. A system as claimed in claim 1 further comprising at least one reference area in which the application files reside.

9. A system as claimed in claim 1 wherein the scanning system comprises a report generator for generating reports from the acyclic dependency graph.

10. A system as claimed in claim 1 further comprising

an information repository for storing and retrieving the acyclic dependency graph;

a plurality of first computer systems including reference areas in which the application files reside and are analyzed by the scanning system;

an second computer system, wherein the scanning system, the information model processor, and the information repository execute on the second computer system; and

a third computer system, wherein the release system executes on the third computer system,

wherein the first computer systems, the second computer system, and the third computer system communicate via a network.

11. A system as in claim 1

wherein the software application comprises source components and derived components, the derived components deriving from the source components, and

wherein the acyclic dependency graph comprises component references and links, the component references comprising source component references and derived component references, and the links comprising derived links and cross-links, the derived links for connecting the source component references and the derived component references, and the cross-links for connecting two derived component references, wherein the two derived component references represent two derived components deriving from the same source component.

12. A method for managing change in a software application, the method comprising the steps of:

scanning application files for determining dependency relationships among the application files, the application files originating from a plurality of software development sources;

processing the dependency relationships to determine an acyclic dependency graph representing the dependency relationships among the application files; and

creating a release version of the software application using the acyclic dependency graph.

13. A method as in claim 12, further comprising the steps of

storing the acyclic dependency graph in an information repository; and

retrieving the acyclic dependency graph from the information repository to create the release version of the software application.

14. A method as in claim 12 further comprising the step of producing the acyclic dependency graph in an output format for use by a release system.

15. A method as in claim 12 wherein the step of scanning application files comprises parsing the application files with at least one parser.

16. A method as in claim 12 wherein the step of scanning application files comprises accessing at least one reference area holding the application files.

17. A method as in claim 12 further comprising the step of generating reports from the acyclic dependency graph.

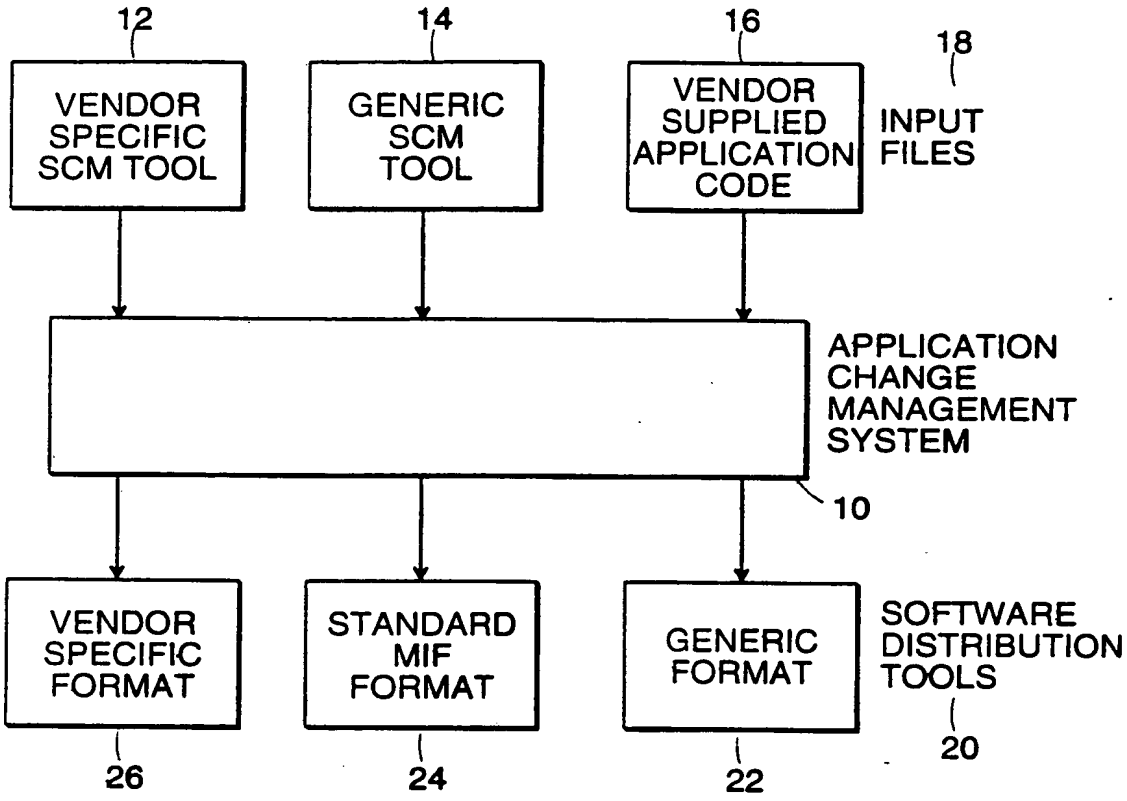


FIG. 1

BNSDOCID: <WO___9857260A1_I_>

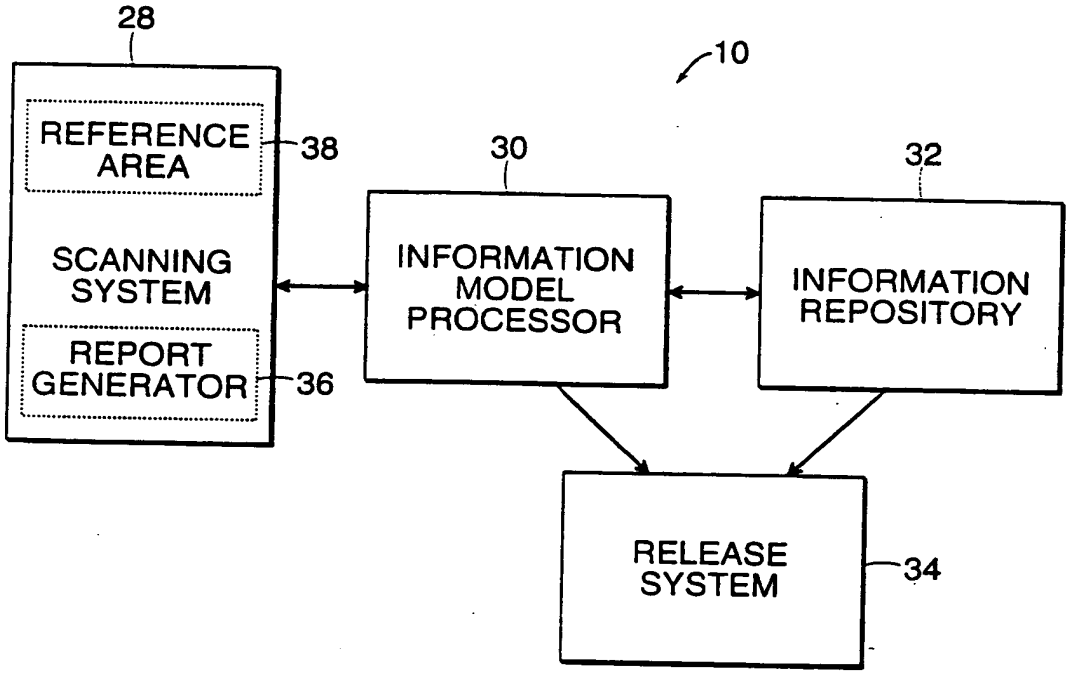

28
10
REFERENCE AREA
38
30
32
SCANNING SYSTEM
INFORMATION MODEL PROCESSOR
INFORMATION REPOSITORY
REPORT GENERATOR
36
RELEASE SYSTEM
34

FIG. 2

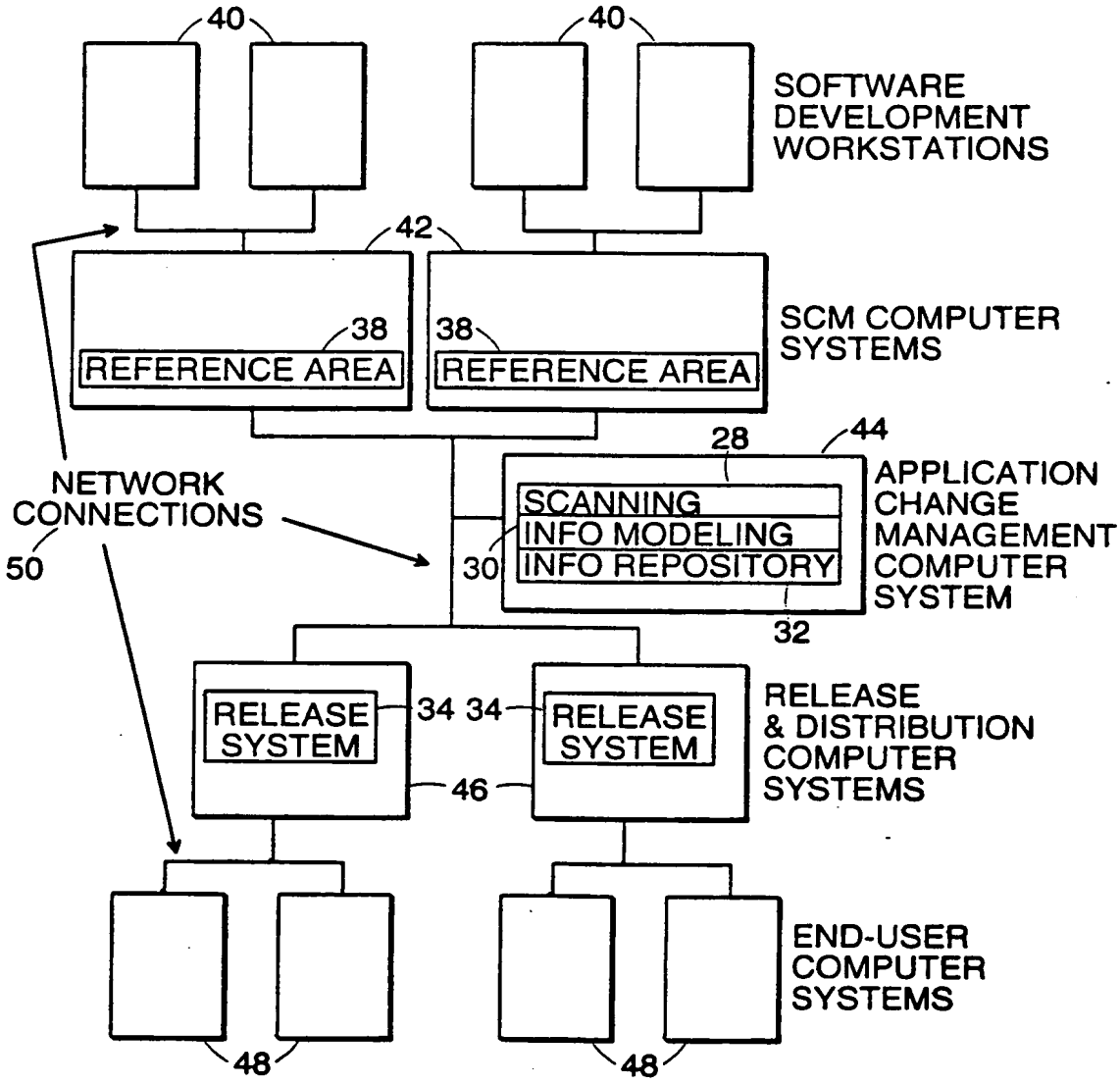


FIG. 3

BNSDOCID: <WO___9857260A1_I_>

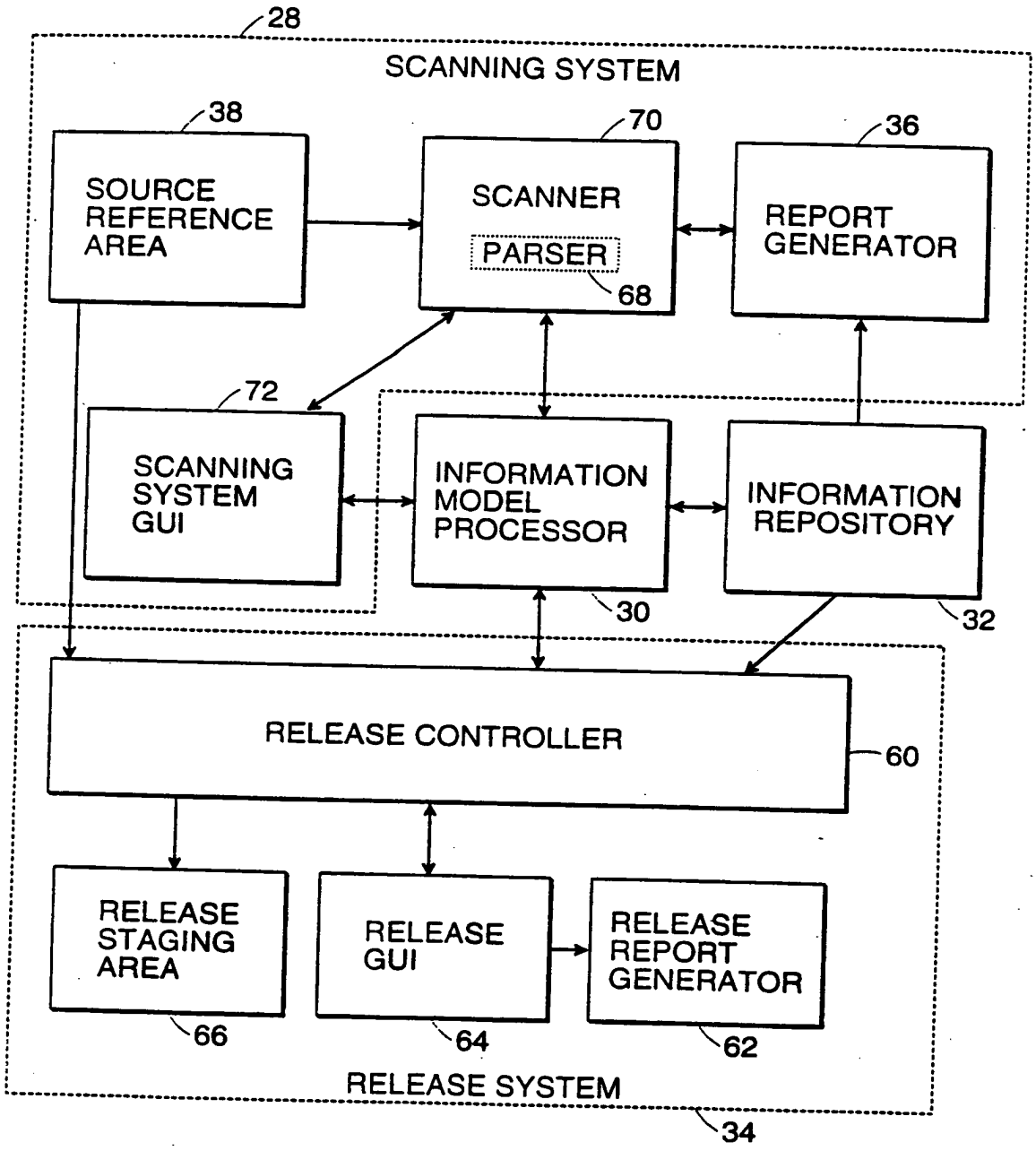

28
SCANNING SYSTEM
38
SOURCE REFERENCE AREA
70
SCANNER
PARSER
68
36
REPORT GENERATOR
72
SCANNING SYSTEM GUI
INFORMATION MODEL PROCESSOR
INFORMATION REPOSITORY
30
32
RELEASE CONTROLLER
60
RELEASE STAGING AREA
66
RELEASE GUI
64
RELEASE REPORT GENERATOR
62
RELEASE SYSTEM
34

FIG. 4

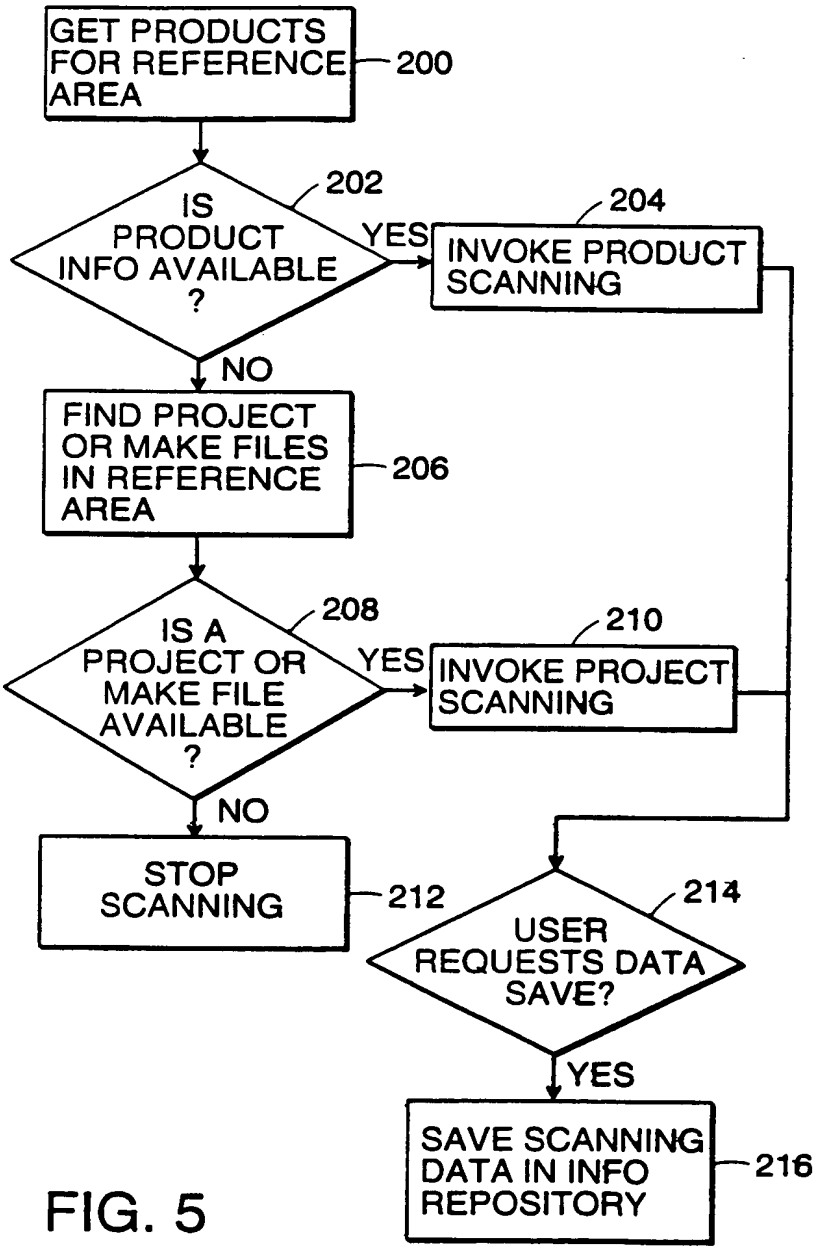


FIG. 5

BNSDOCID: <WO___9857260A1_I_>

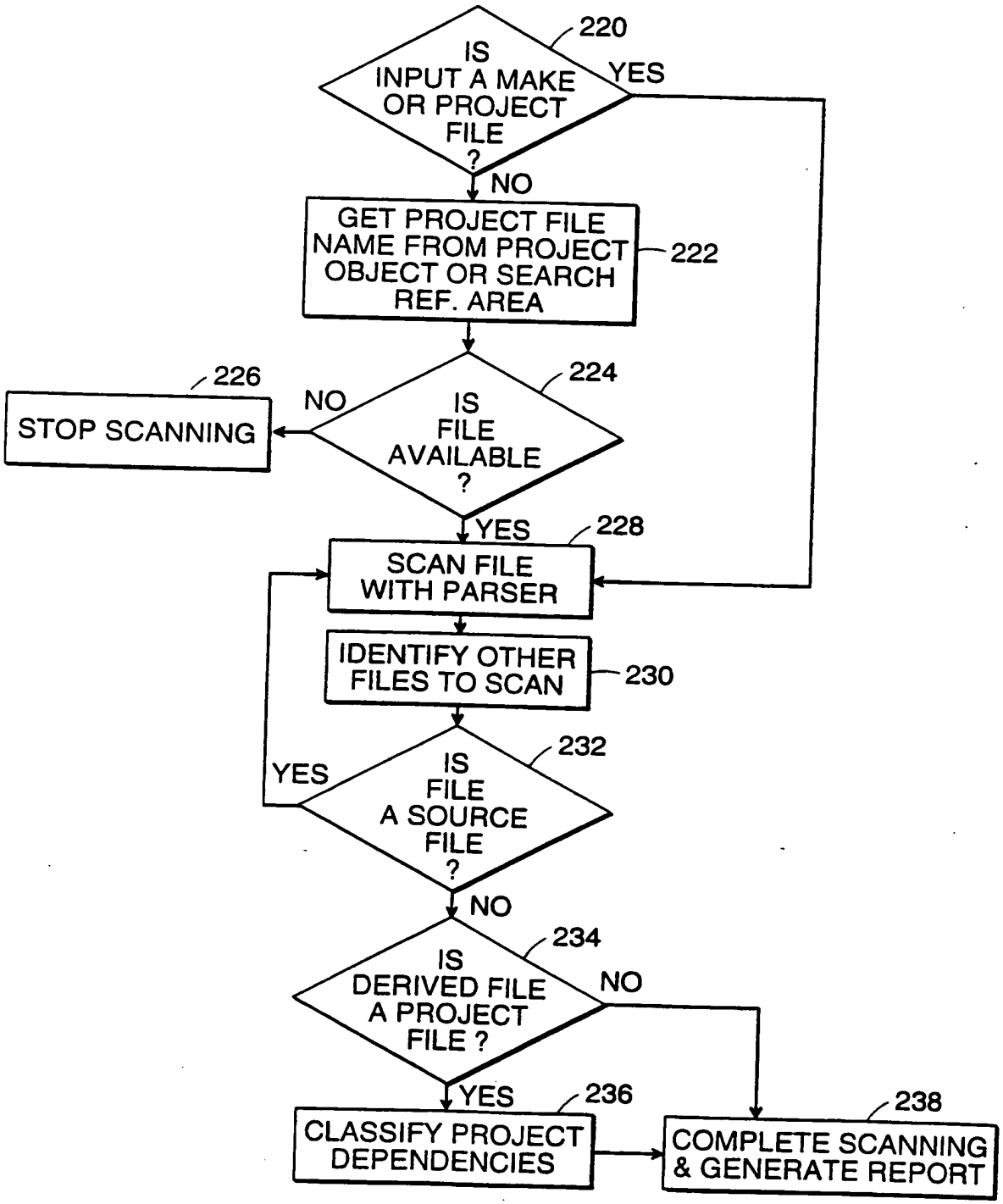

220
IS INPUT A MAKE OR PROJECT FILE ?
YES
NO
GET PROJECT FILE NAME FROM PROJECT OBJECT OR SEARCH REF. AREA
222
224
IS FILE AVAILABLE ?
NO
226
STOP SCANNING
YES
228
SCAN FILE WITH PARSER
IDENTIFY OTHER FILES TO SCAN
230
232
IS FILE A SOURCE FILE ?
YES
NO
234
IS DERIVED FILE A PROJECT FILE ?
NO
YES
236
CLASSIFY PROJECT DEPENDENCIES
238
COMPLETE SCANNING & GENERATE REPORT

FIG. 6

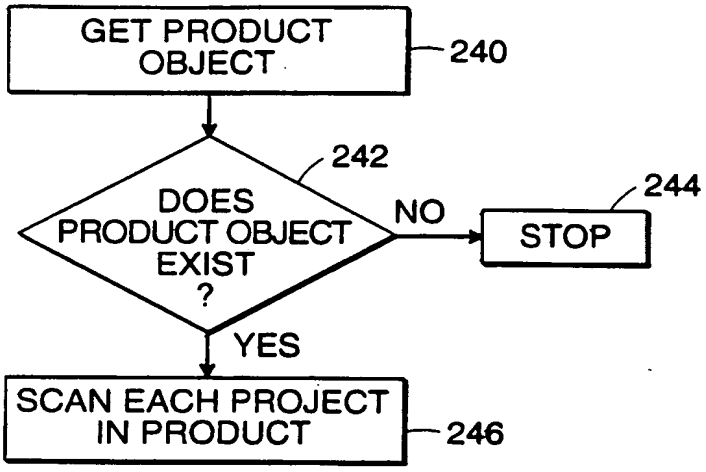


FIG. 7

BNSDOCID: <WO___9857260A1_I_>

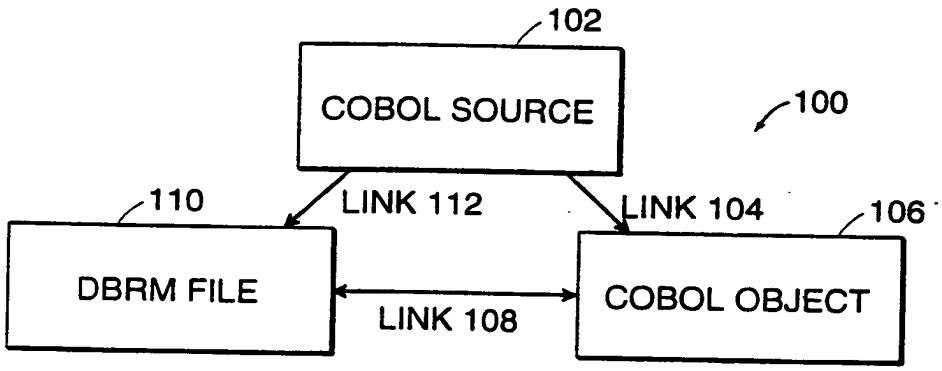

102
COBOL SOURCE
100
110
LINK 112
LINK 104
106
DBRM FILE
LINK 108
COBOL OBJECT

FIG. 8

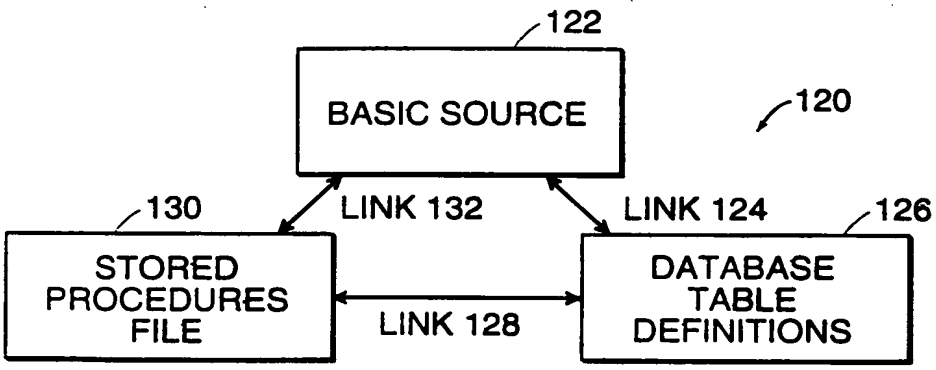


FIG. 9

BNSDOCID: <WO___9857260A1_I_>

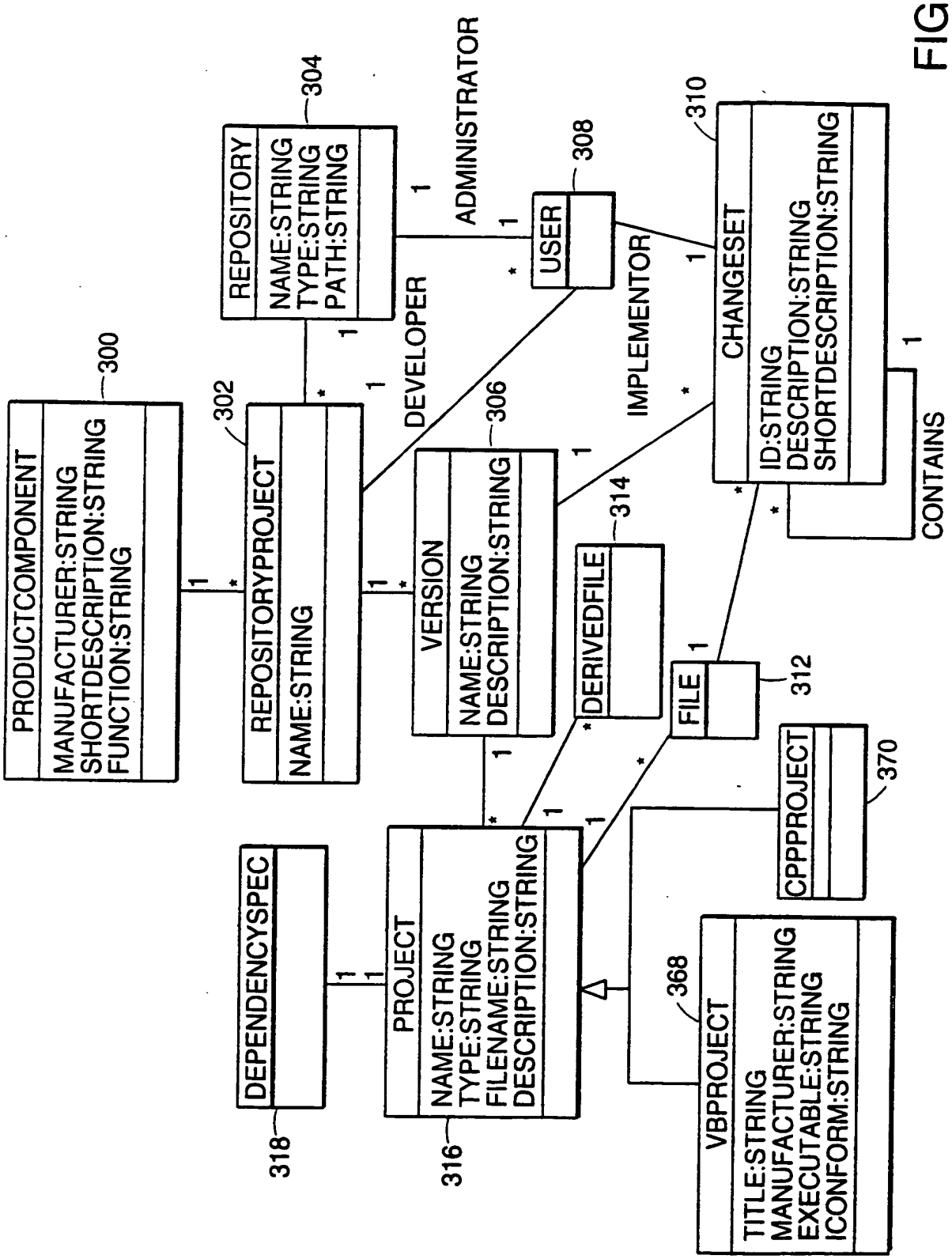
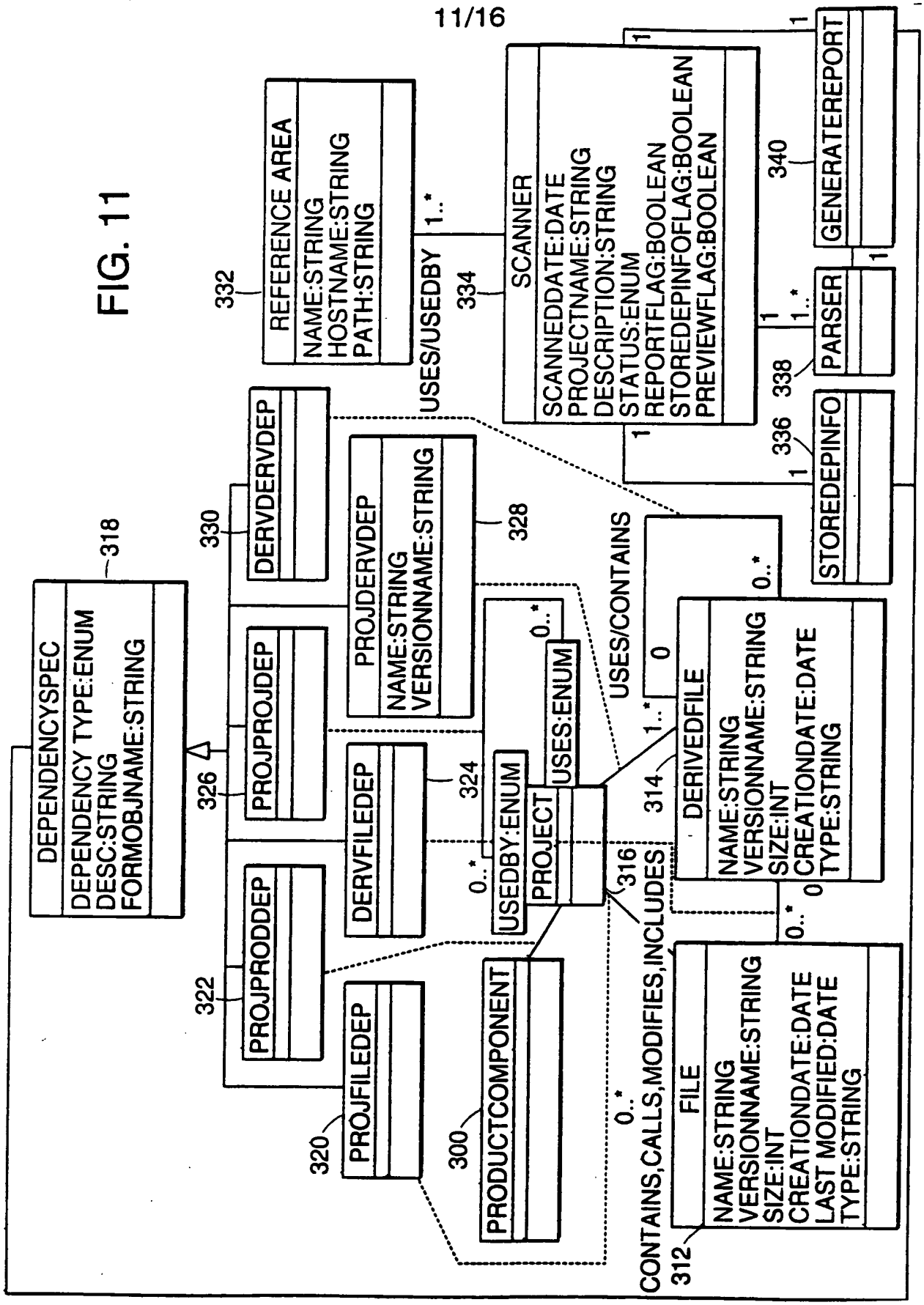


FIG. 10

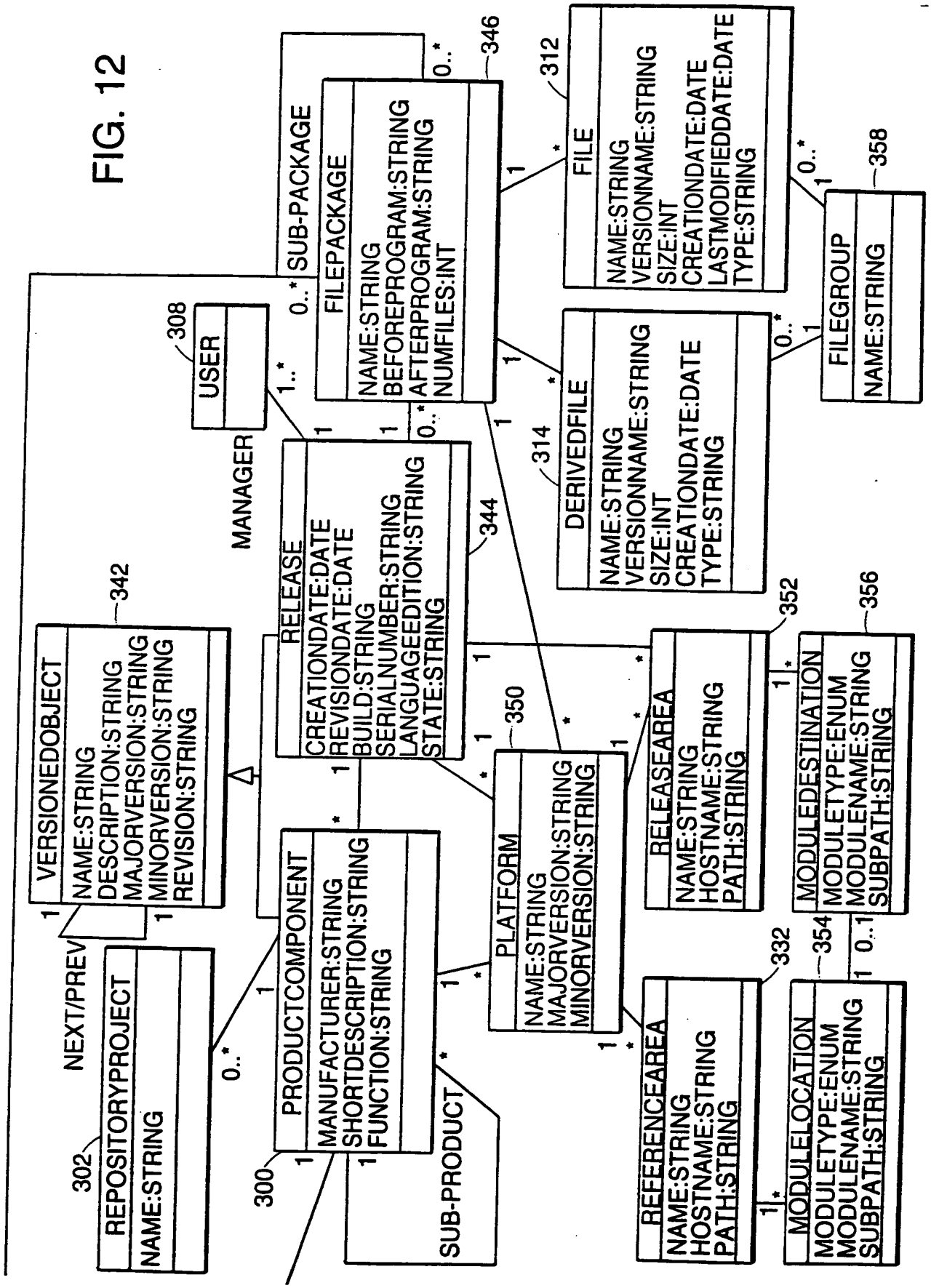
BNSDOCID: <WO___9857260A1_I_>

FIG. 11



BNSDOCID: <WO___9857260A1_I_>

FIG. 12



BNSDOCID: <WO___9857260A1_I_>

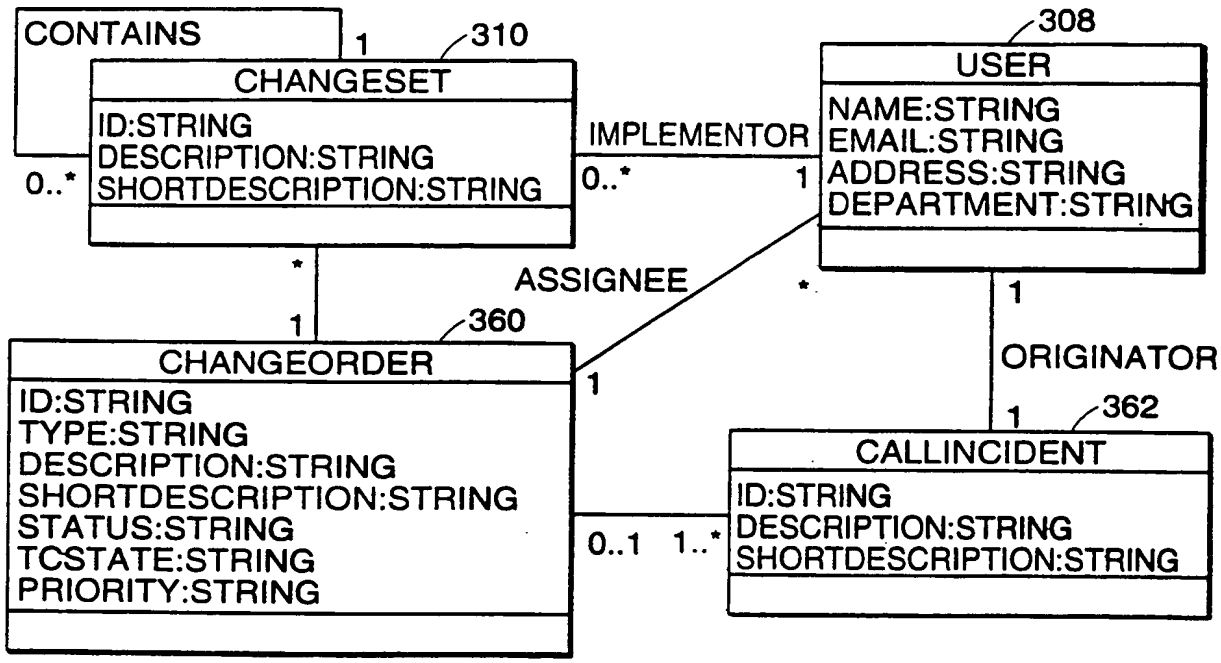


FIG. 13

BNSDOCID: <WO___9857260A1_I_>

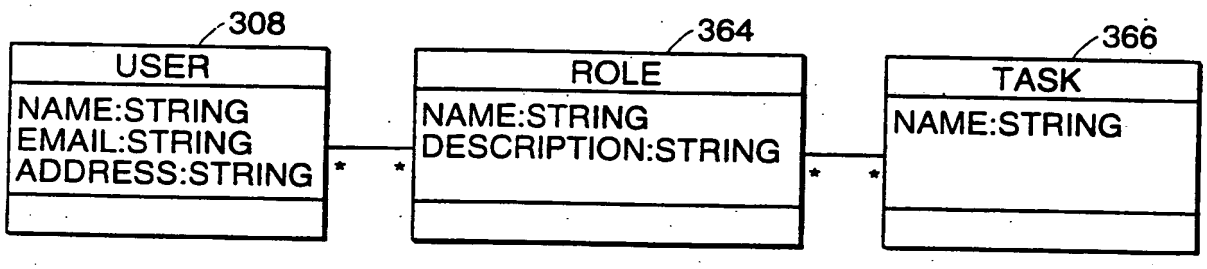

308
USER
NAME:STRING
EMAIL:STRING
ADDRESS:STRING
364
ROLE
NAME:STRING
DESCRIPTION:STRING
366
TASK
NAME:STRING
*
*
*
*

FIG. 14

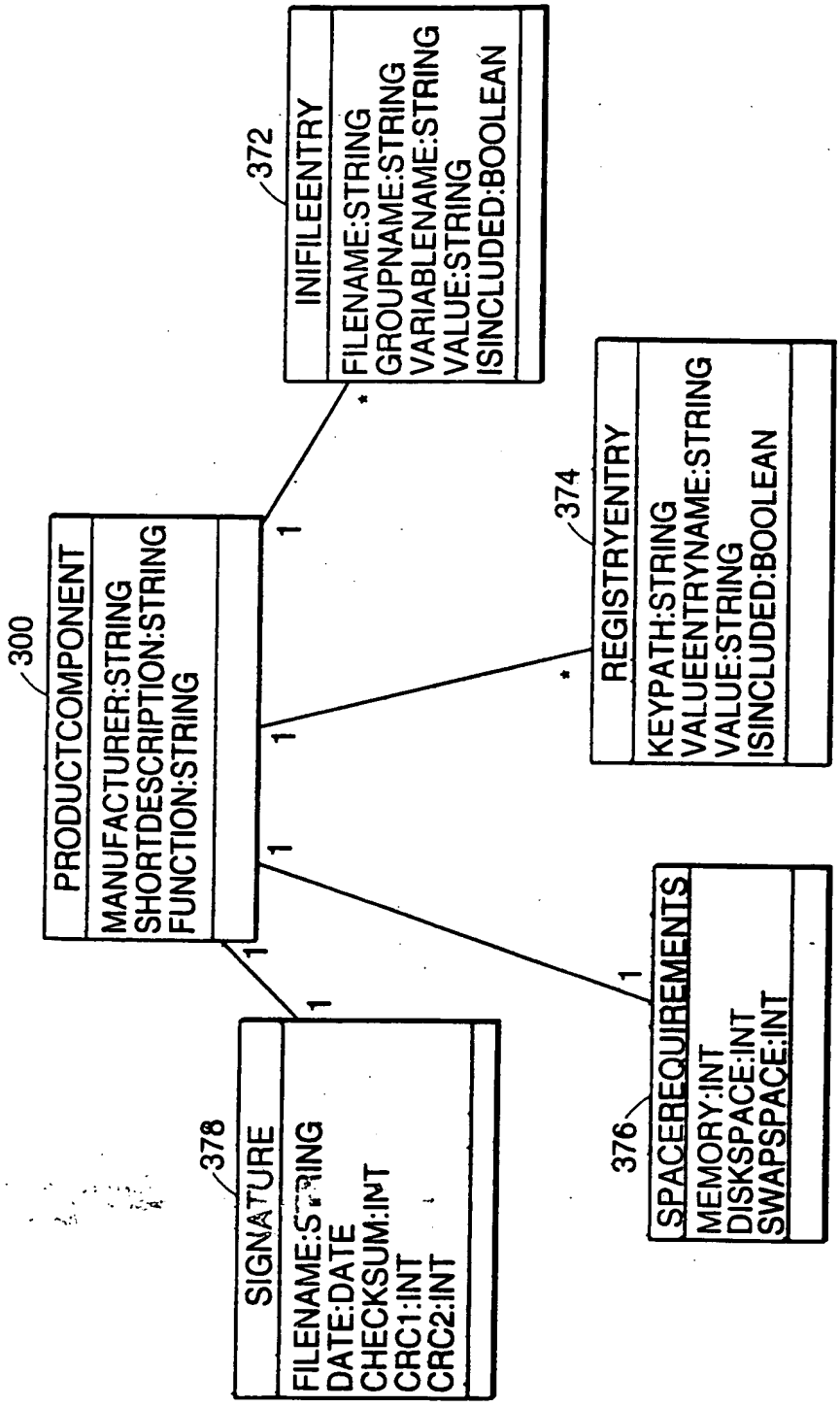


FIG. 15

BNSDOCID: <WO___9857260A1_I_>

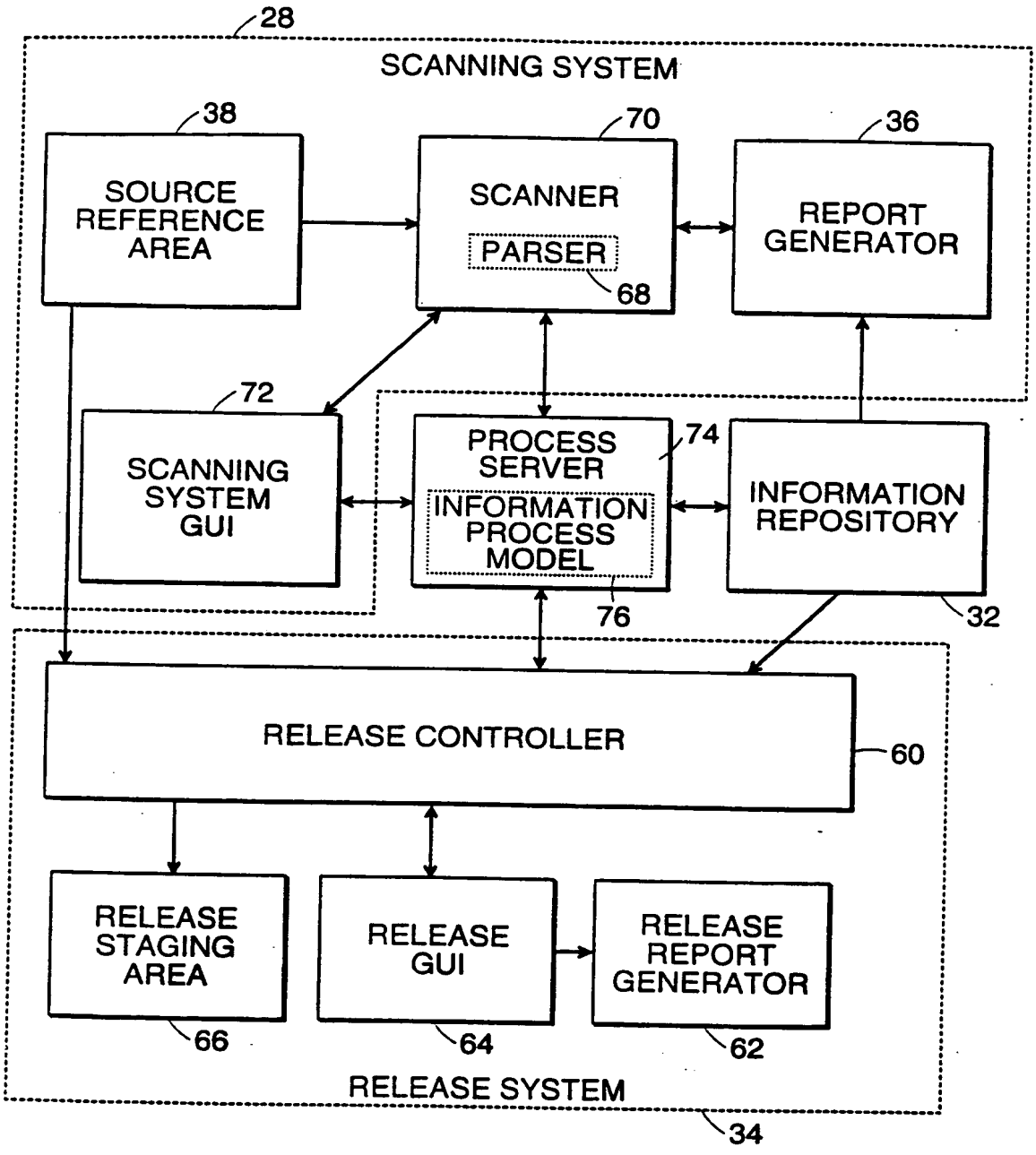


FIG. 16

BNSDOCID: <WO___9857260A1_I_>

INTERNATIONAL SEARCH REPORT

Interr ial Application No
PCT/US 98/12050

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 496 494 A (IBM) 29 July 1992	1,3-8, 10,12-15
Y	see column 1, line 1 - column 2, line 35 see column 6, line 18 - line 45	2,8
Y	EP 0 501 613 A (HEWLETT PACKARD CO) 2 September 1992	2
A	see abstract; claims 1-3,5,6,8; figures 1,2	10
Y	US 5 361 357 A (KIONKA DANIEL P) 1 November 1994 see abstract see column 1, line 1 - line 67	8

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
18 September 1998	25/09/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Kingma, Y

1

Form PCT/ISA/210 (second sheet) (July 1992)

BNSDOCID: <WO___9857260A1_I_>

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter nal Application No
PCT/US 98/12050

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0496494 A	29-07-1992	CA 2059577 A	23-07-1992
		JP 2066376 C	24-06-1996
		JP 5061683 A	12-03-1993
		JP 7092748 B	09-10-1995
		US 5493682 A	20-02-1996
EP 0501613 A	02-09-1992	JP 5197697 A	06-08-1993
		US 5339435 A	16-08-1994
US 5361357 A	01-11-1994	NONE	

Form PCT/ISA/210 (patent family annex) (July 1992)

BNSDOCID: <WO___9857260A1_I_>

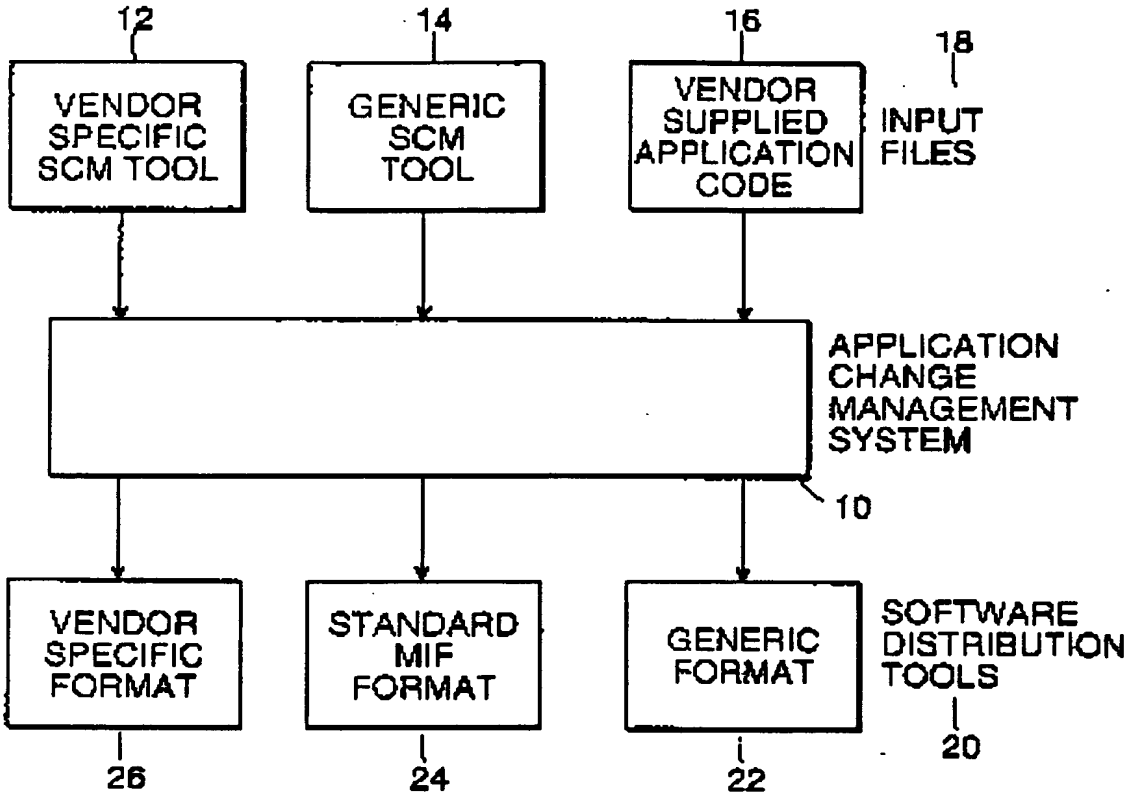


FIG. 1

BNSDOCID: <WO___9857260A1TI_>

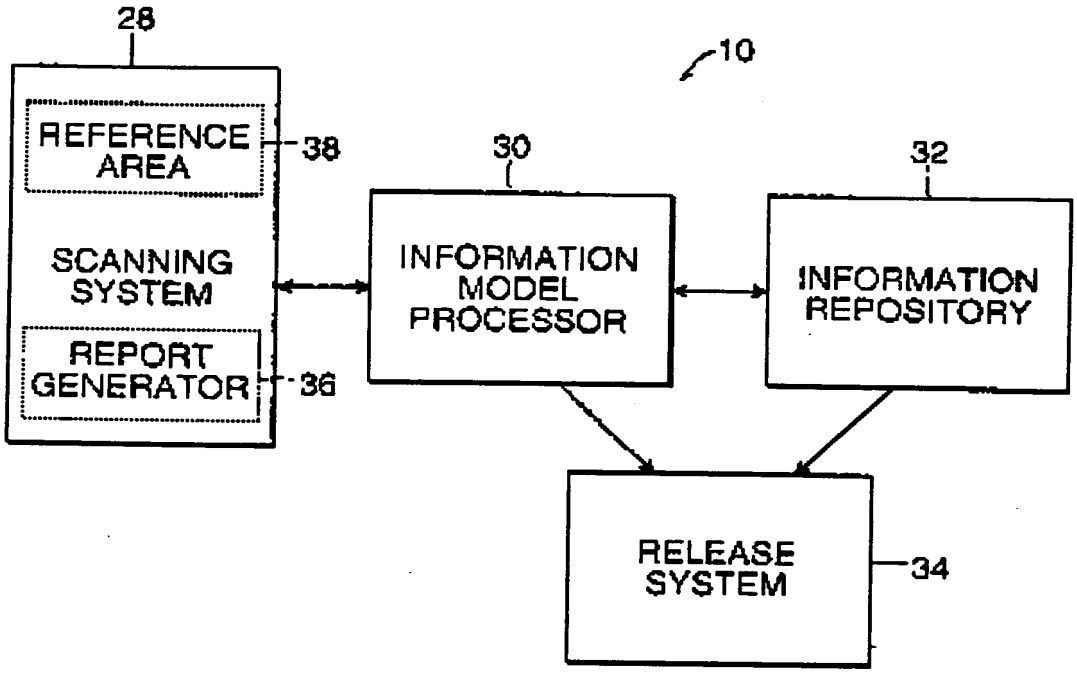

28
10
REFERENCE AREA
38
30
32
SCANNING SYSTEM
INFORMATION MODEL PROCESSOR
INFORMATION REPOSITORY
REPORT GENERATOR
36
RELEASE SYSTEM
34

FIG. 2

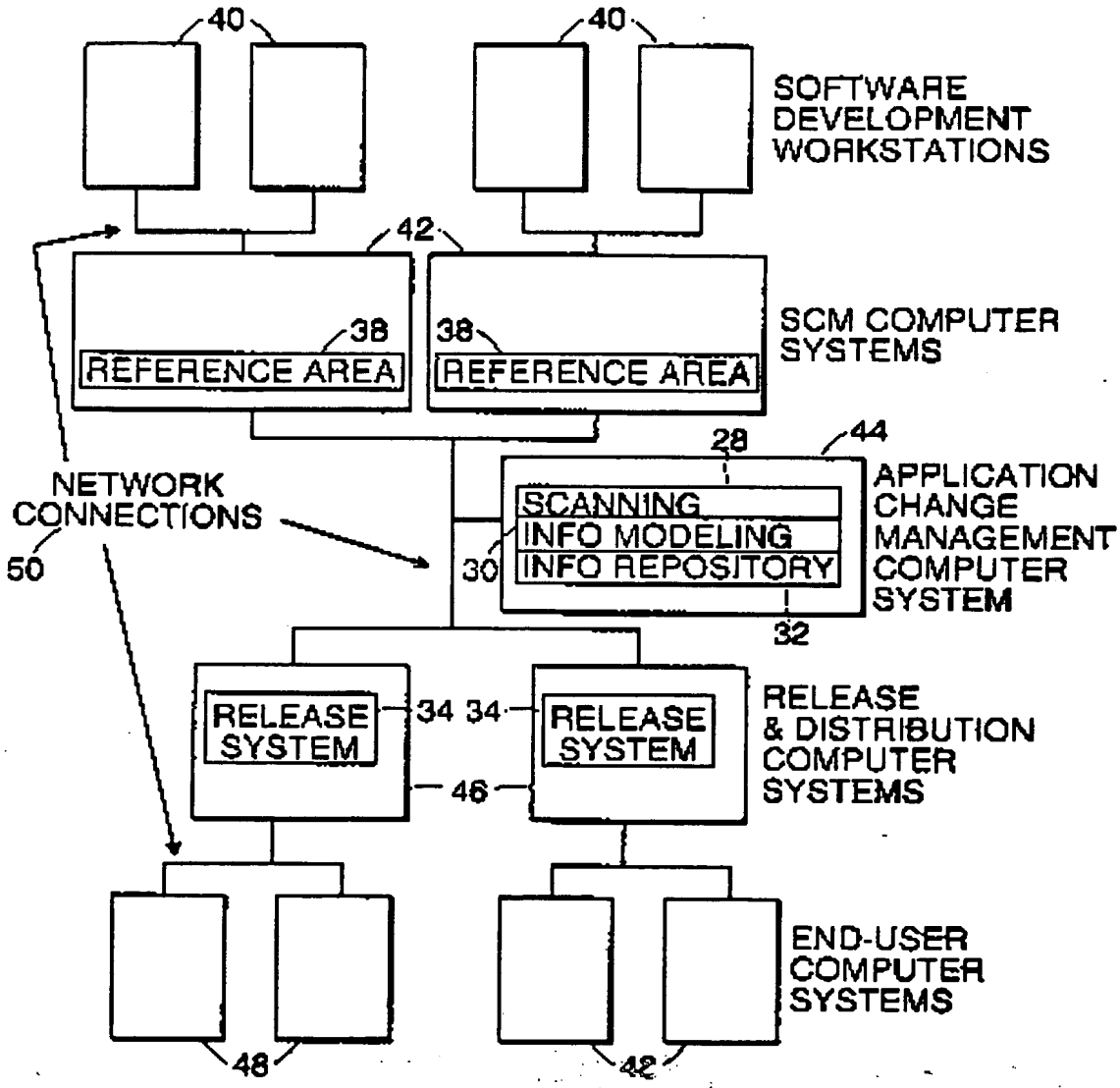


FIG. 3

BNSDOCID: <WO__9857260A1T1_>

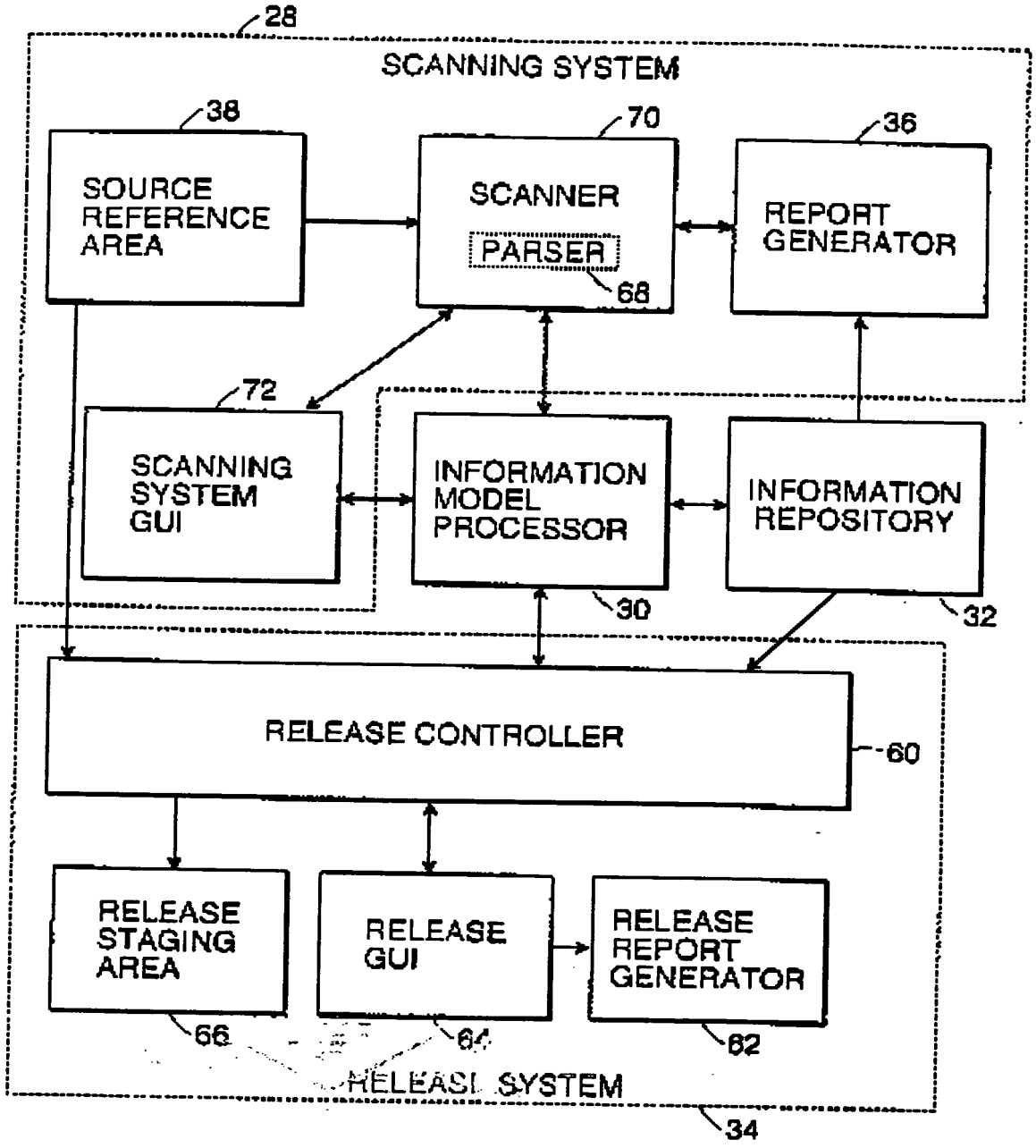

28
SCANNING SYSTEM
38
SOURCE REFERENCE AREA
70
SCANNER
PARSER
68
36
REPORT GENERATOR
72
SCANNING SYSTEM GUI
INFORMATION MODEL PROCESSOR
30
INFORMATION REPOSITORY
32
RELEASE CONTROLLER
60
RELEASE STAGING AREA
66
RELEASE GUI
64
RELEASE REPORT GENERATOR
62
RELEASE SYSTEM
34

FIG. 4

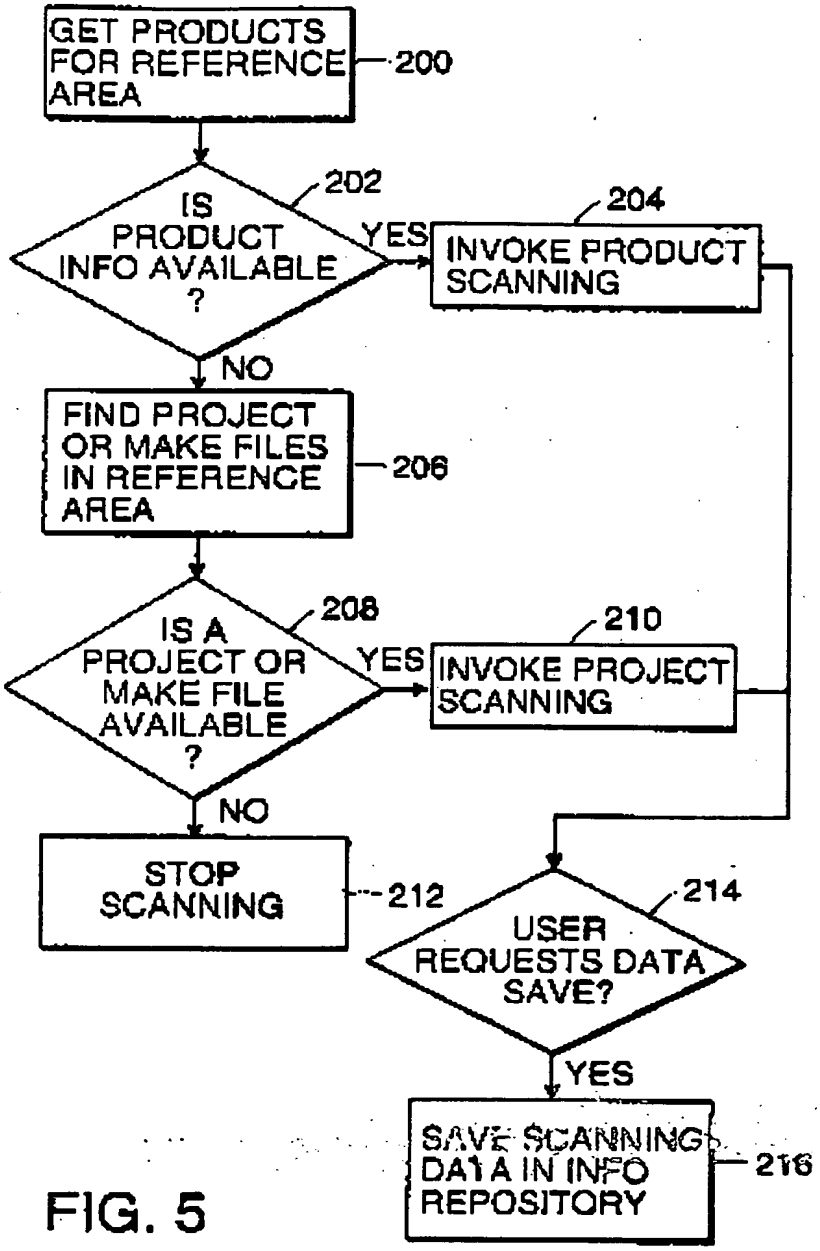


FIG. 5

BNSDOCID: <WO___9857260A1T1_>

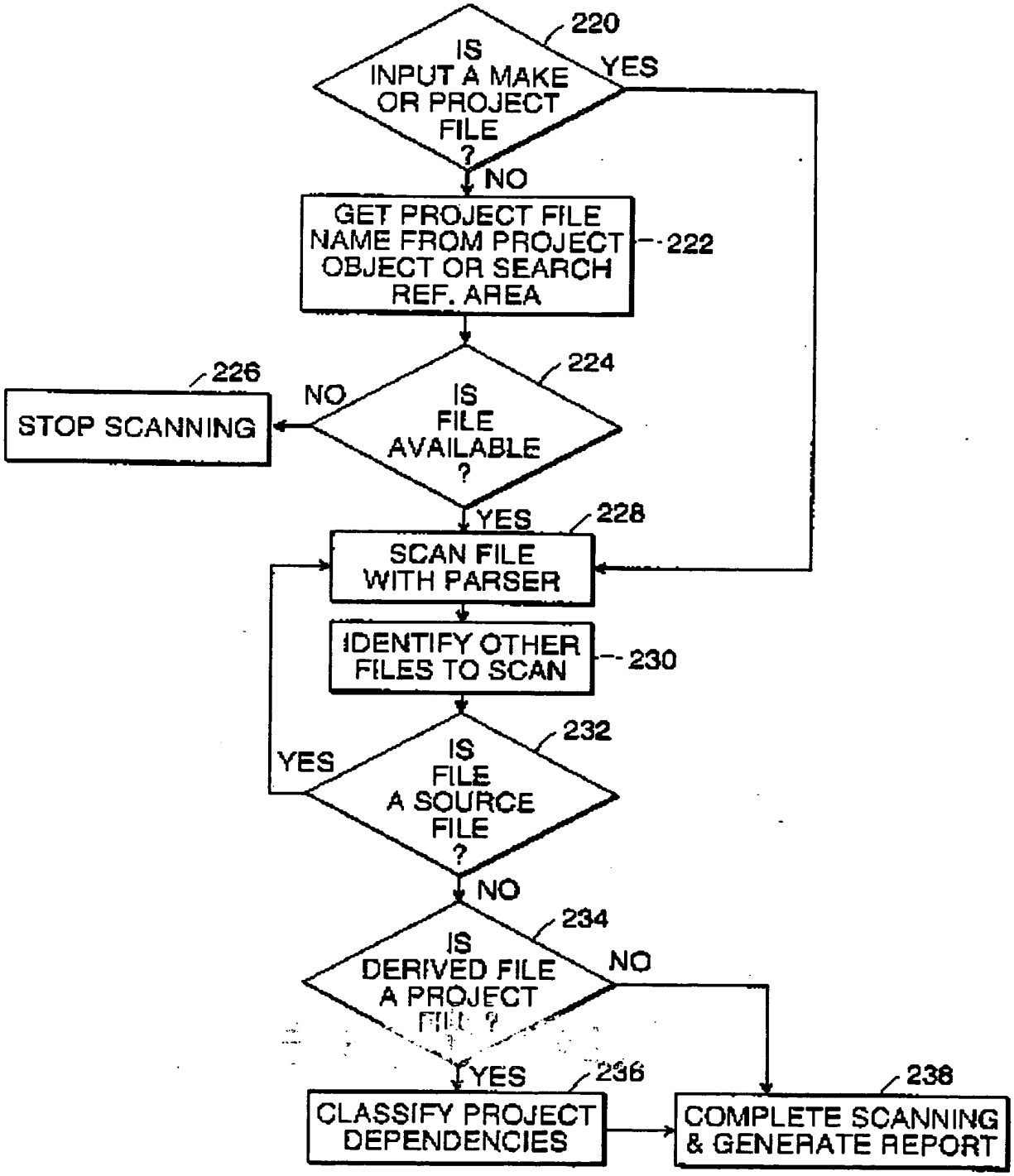


FIG. 6

BNSDOCID: <WO___9857260A1T1_>

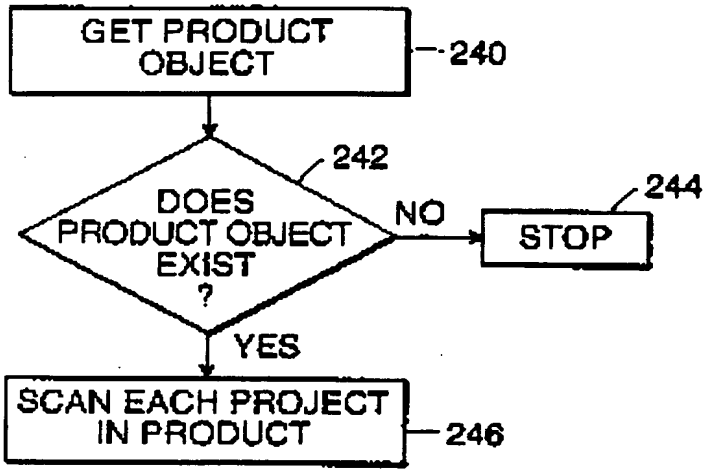


FIG. 7

BNSDOCID: <WO__9857260A1TI_>

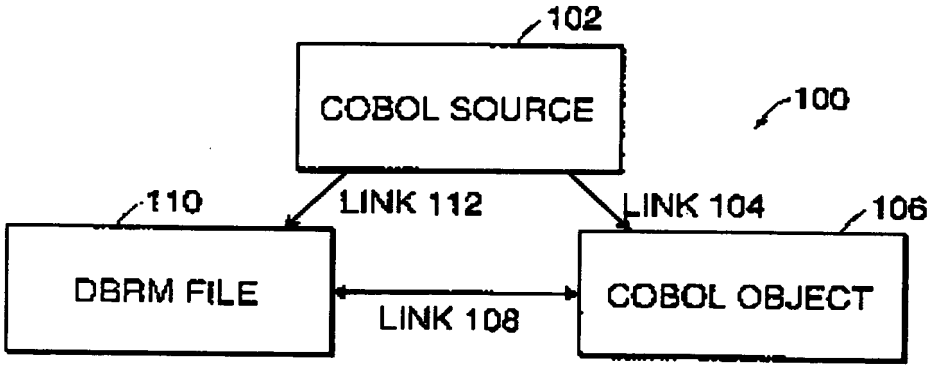

102
COBOL SOURCE
100
110
LINK 112
LINK 104
106
DBRM FILE
LINK 108
COBOL OBJECT

FIG. 8

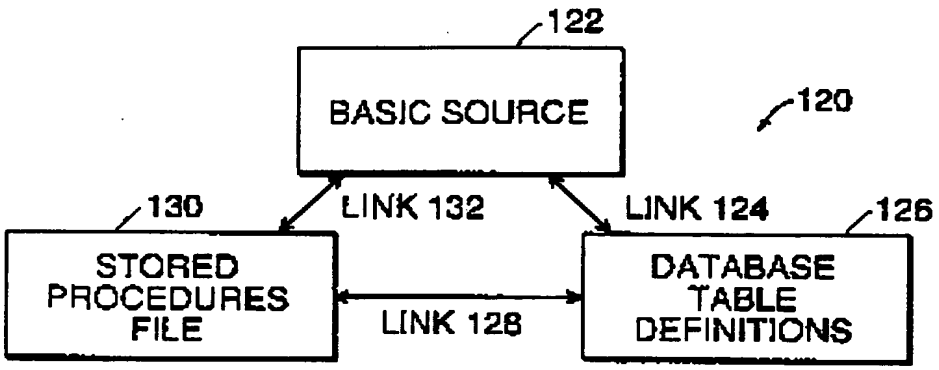


FIG. 9

BNSDOCID: <WO__9857260A1T1_>

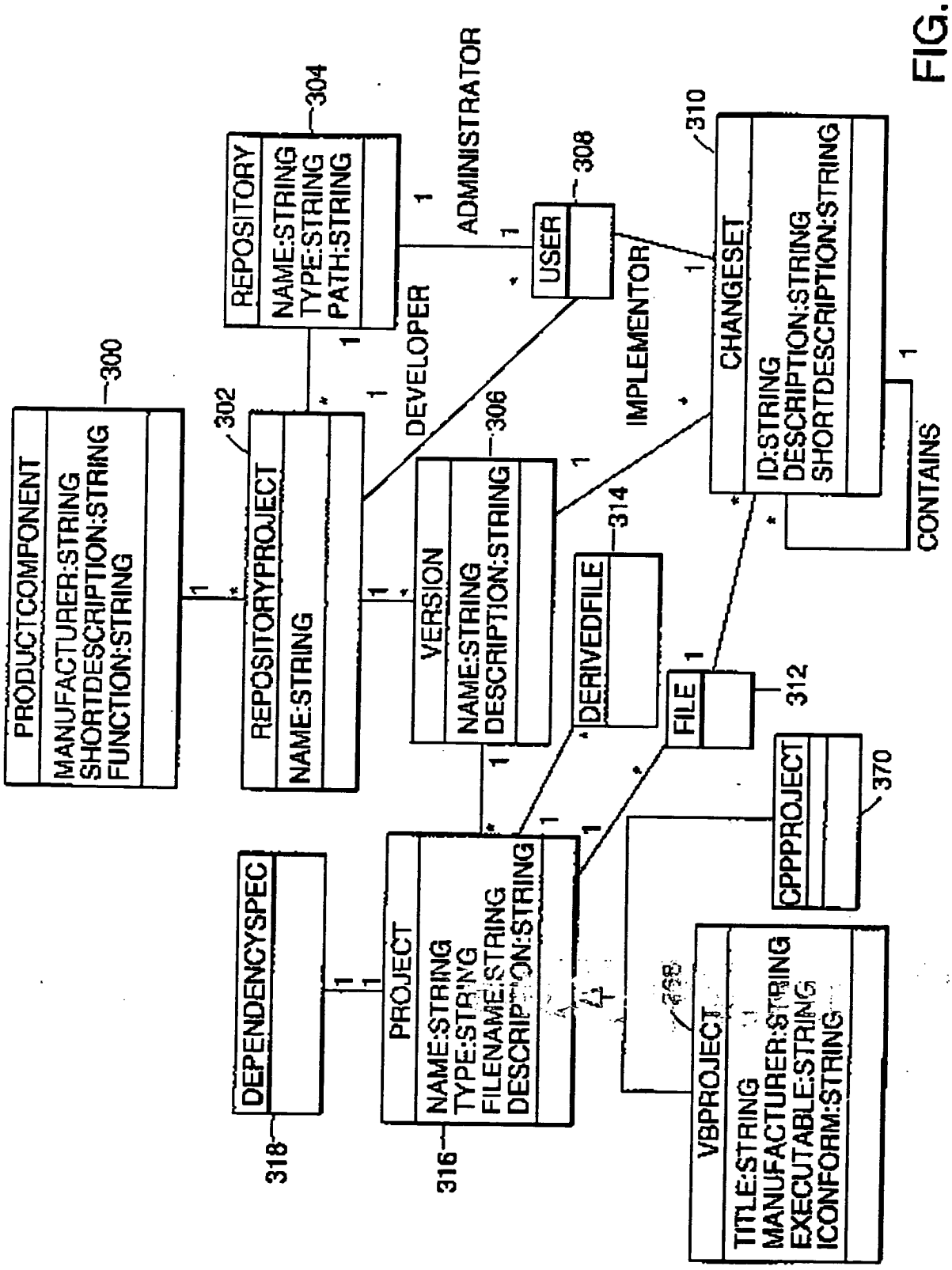
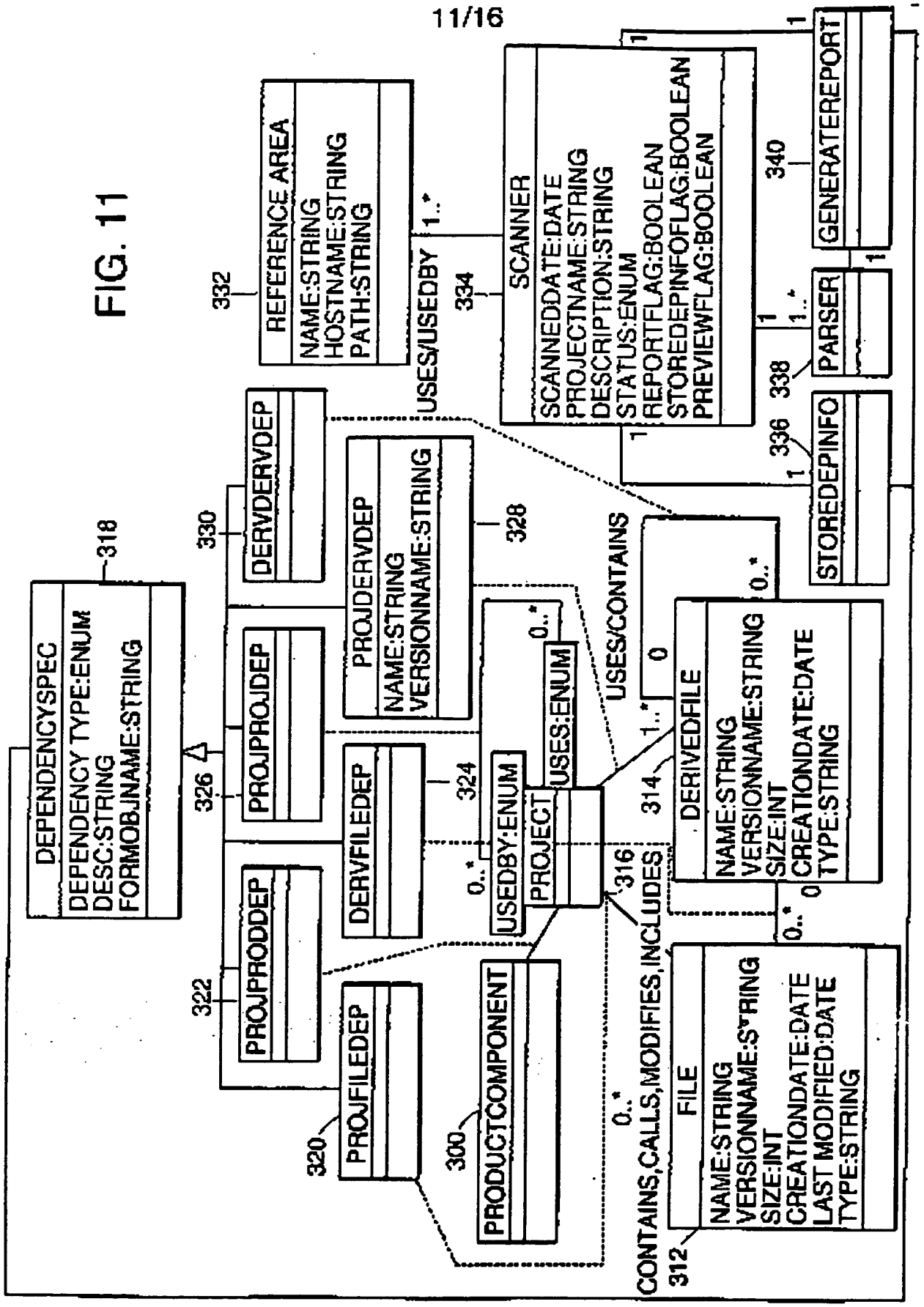


FIG. 10

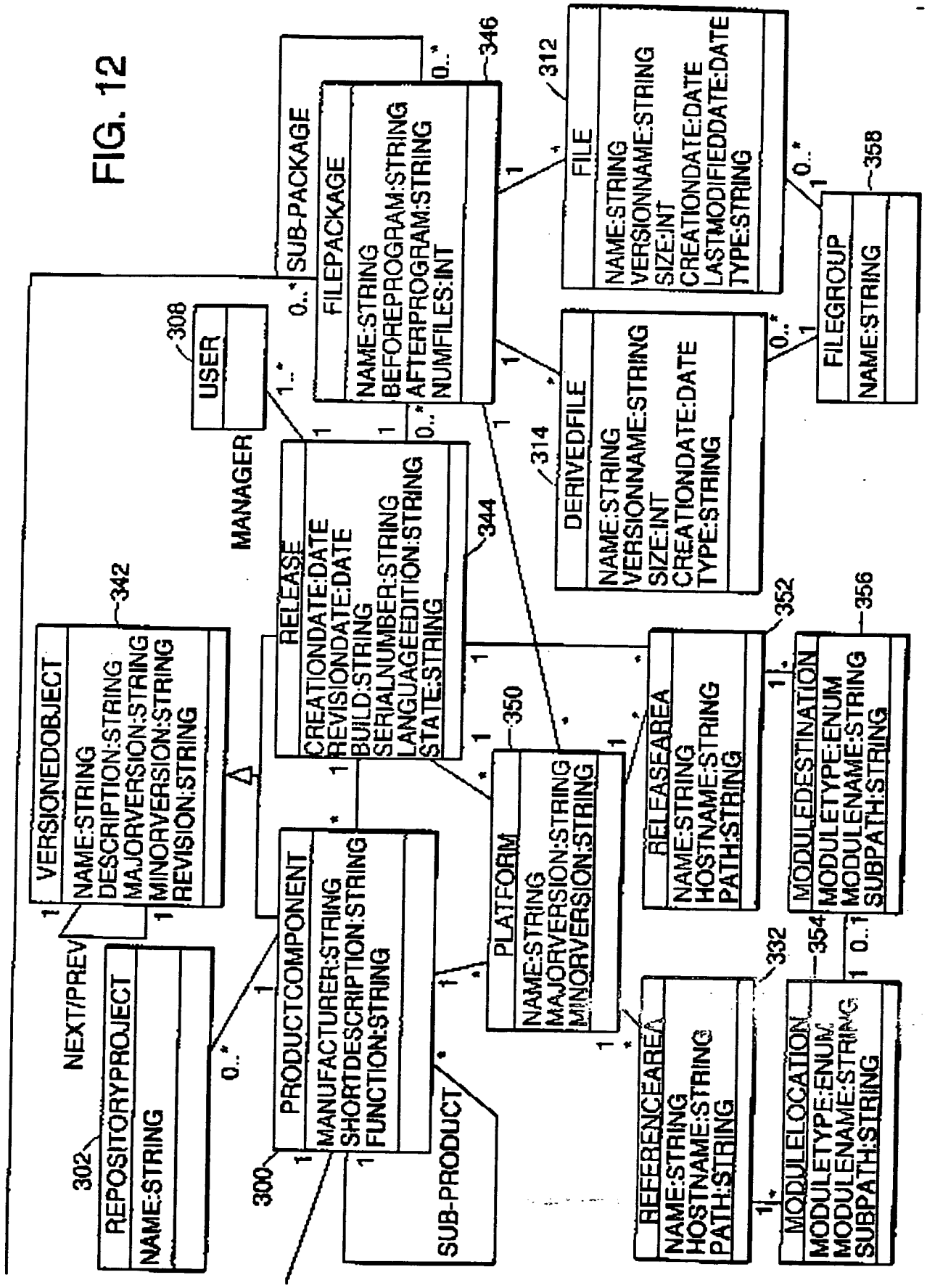
BNSDOCID: <WO___9857260A1TI_>

FIG. 11



BNSDOCID: <WO___9857260A1T1_>

FIG. 12



BNSDOCID: <WO___9857260A1TI:>

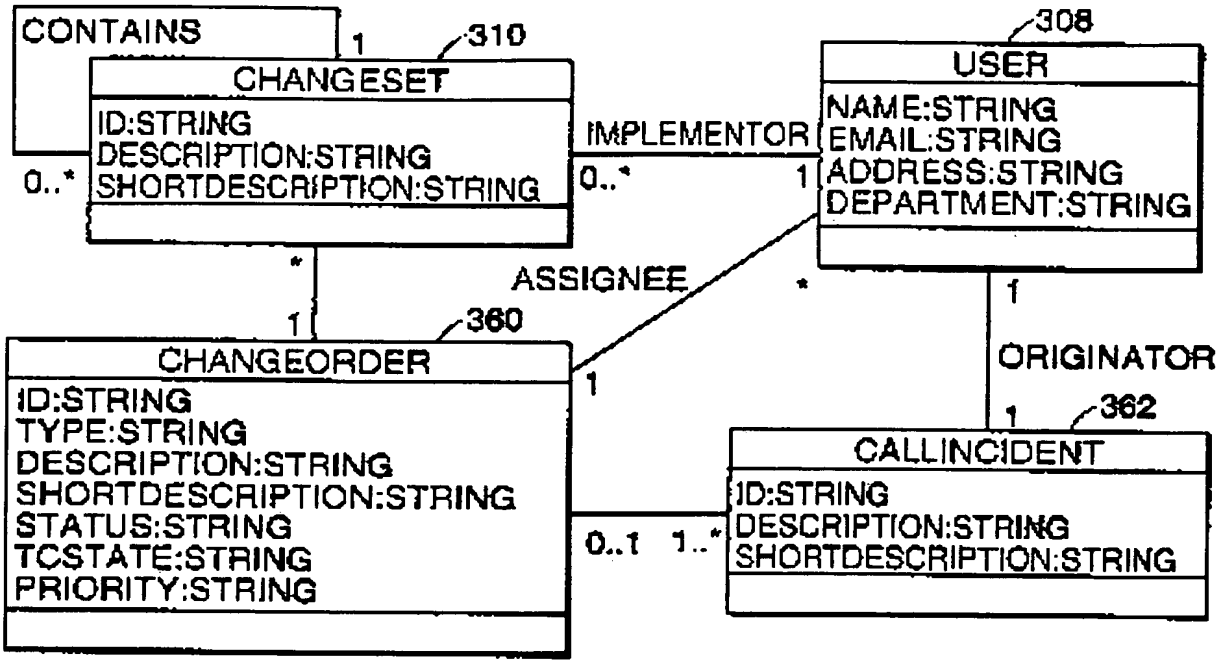


FIG. 13

BNSDOCID: <WO___9857260A1TI_>

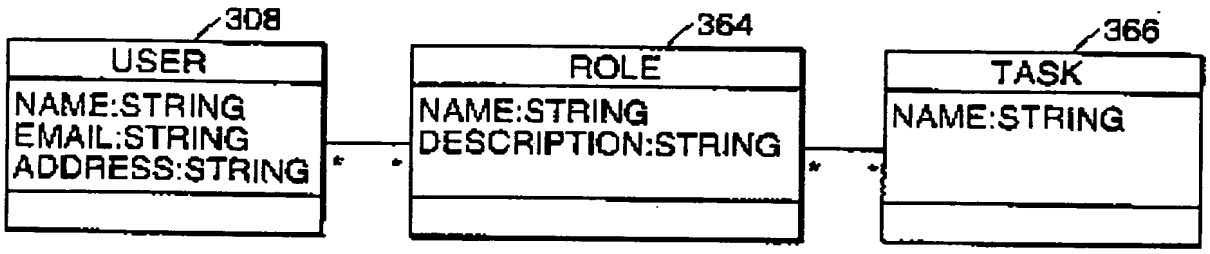


FIG. 14

BNSDOCID: <WO___9857260A1TI_>

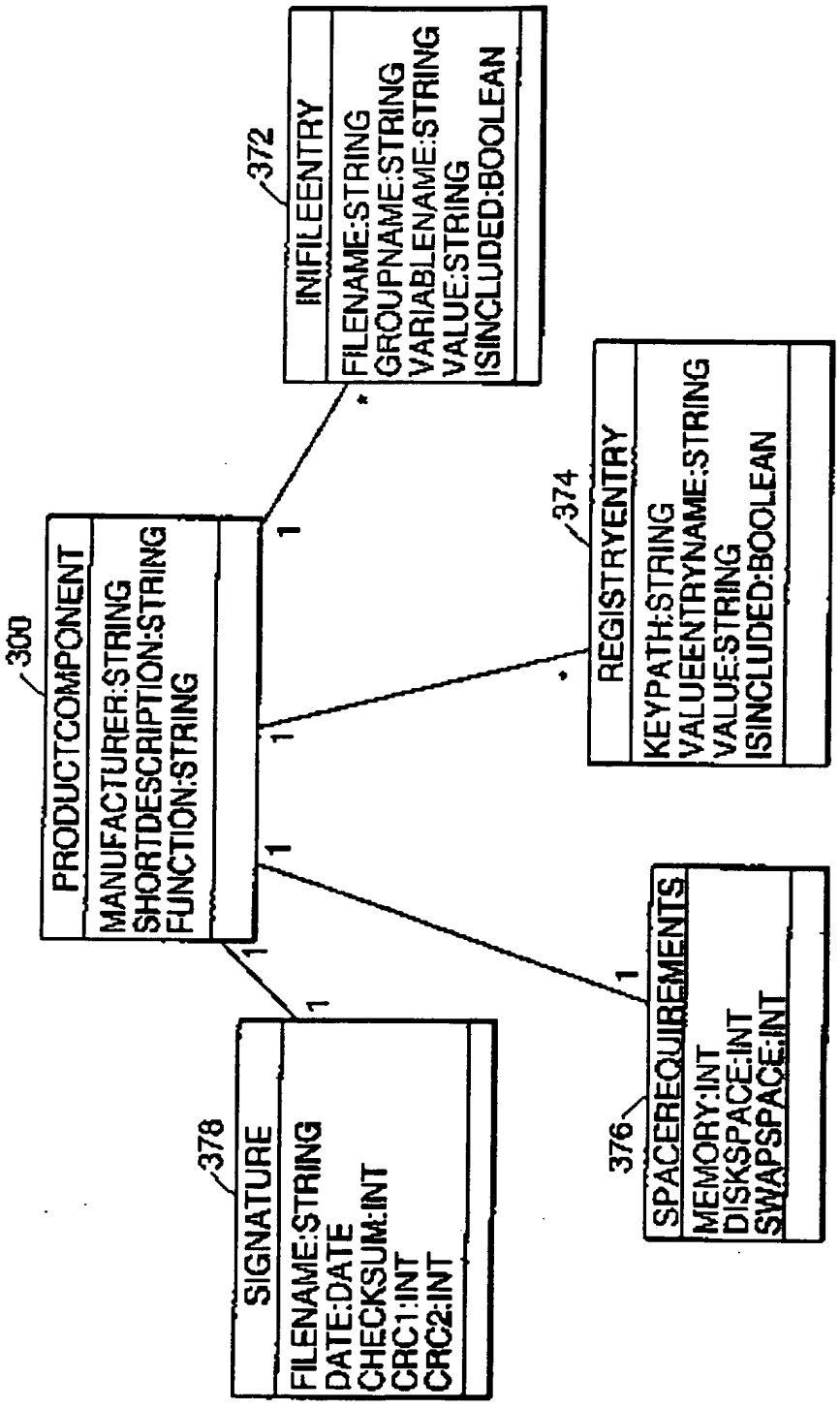


FIG. 15

BNSDOCID: <WO___9857260A1T1_>

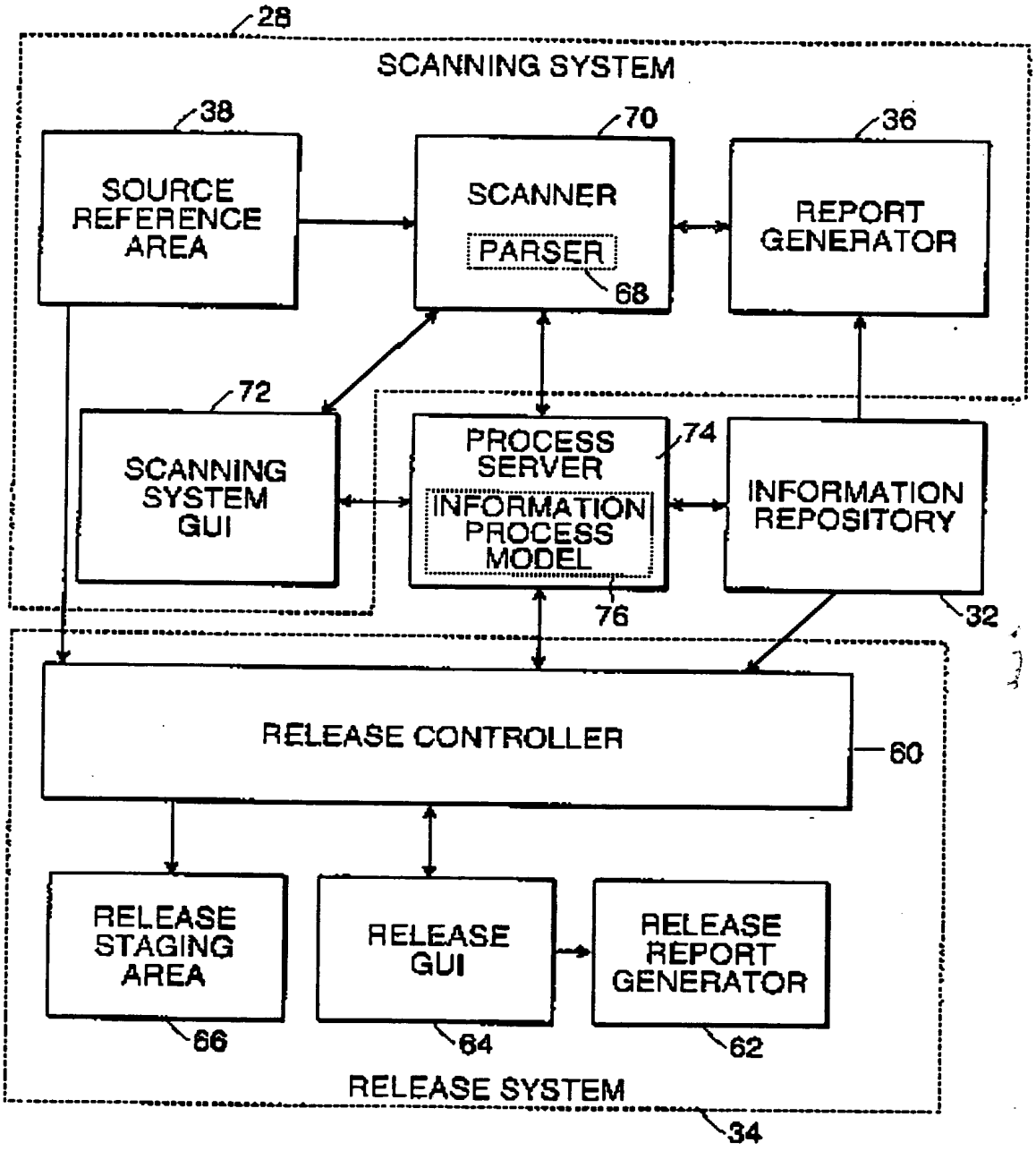

28
SCANNING SYSTEM
38
SOURCE REFERENCE AREA
70
SCANNER
PARSER
68
36
REPORT GENERATOR
72
SCANNING SYSTEM GUI
PROCESS SERVER
INFORMATION PROCESS MODEL
74
INFORMATION REPOSITORY
76
32
RELEASE CONTROLLER
60
RELEASE STAGING AREA
66
RELEASE GUI
64
RELEASE REPORT GENERATOR
62
RELEASE SYSTEM
34

FIG. 16

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

THIS PAGE BLANK (USPTO)